Question - 1

MCQ 1

SCORE: **1 points**

Consider the HTML file shown below and the corresponding CSS rules (**for simplicity, all 3 MCQ questions use the same file**).

In the DOM tree, which of the following nodes will be selected using the `querySelectorAll` with the following CSS selector string " `#three>.B` " ?

```
<html>

<head>
   <style>
       #one { color: red; }              /* Rule #1 */
       .A { color: blue; }               /* Rule #2 */
       .A .B { color: green; }           /* Rule #3 */
       .A #five { color: yellow; }       /* Rule #4 */
   </style>
</head>

<body>
   <div id="one" class="A">
       <h1> This </h1>
       <div id="two" class="B">
           <h2> is </h2>
           <div id="three" class="A">
               <h3 id="six"> a </h3>
               <div id="four" class="B">
                   <p id="five">
                       test webpage.
                   </p>
               </div>
           </div>
       </div>
   </div>
</body>

</html>
```

- <p id="five">
- <h3 id="six">
-
- None of the above

Question - 2

MCQ 2

SCORE: **1 points**

Consider the HTML file shown below and the corresponding CSS rules (**for simplicity, all 3 MCQ questions use the same file**).

If you wanted to change the color of the `h3` element to green without any modifications of the CSS rules and only modifying the HTML code, what should you do?

```
<html>

<head>
   <style>
       #one { color: red; }              /* Rule #1 */
       .A { color: blue; }               /* Rule #2 */
       .A .B { color: green; }           /* Rule #3 */
       .A #five { color: yellow; }       /* Rule #4 */
```

```
   </style>
</head>

<body>
   <div id="one" class="A">
       <h1> This </h1>
       <div id="two" class="B">
           <h2> is </h2>
           <div id="three" class="A">
               <h3 id="six"> a </h3>
               <div id="four" class="B">
                   <p id="five">
                       test webpage.
                   </p>
               </div>
           </div>
       </div>
   </div>
</body>

</html>
```

- ( ) Nothing
- (•) Set the class of h3 element to B
- ( ) Set the ID of the h3 element to "five"
- (•) Change the class of the div element with id="three" to B.

Question - 3

MCQ 3

SCORE: **1 points**

Consider the HTML file shown below and the corresponding CSS rules (**for simplicity, all 3 MCQ questions use the same file**).

Assume that you add a button as a child of the `div` element with `id='three'`, and you click on the button. However, the button does not have a handler registered for the click event. However, all the `div` have click handlers for the capture and bubble events. In the following, the prefix C denotes a Capture handler, and the prefix B denotes a Bubble handler. Which of the following is the correct order in which the handlers are called ?

```
<html>

<head>
   <style>
       #one { color: red; }              /* Rule #1 */
       .A { color: blue; }               /* Rule #2 */
       .A .B { color: green; }           /* Rule #3 */
       .A #five { color: yellow; }       /* Rule #4 */
   </style>
</head>

<body>
   <div id="one" class="A">
       <h1> This </h1>
       <div id="two" class="B">
           <h2> is </h2>
           <div id="three" class="A">
               <h3 id="six"> a </h3>
               <div id="four" class="B">
                   <p id="five">
                       test webpage.
                   </p>
               </div>
           </div>
       </div>
   </div>
</body>

</html>
```

- [ ] C-one, C-two, C-three, C-four, C-five, B-five, B-four, B-three, B-two, B-one
- [ ] C-one, C-two, C-three, C-four, B-four, B-three, B-two, B-one
- [x] C-one, C-two, C-three, B-three, B-two, B-one
- [ ] C-one, C-two, B-two, B-one

Question - 4
MCQ 4

SCORE: **1 points**

Assume that you have a function `foo` that is sending an AJAX message using `xhr.send`. The xhr object's `onload` has been initialized to `goo` before the `send` call. The server is running on the same machine, say, and hence has near zero latency. The response from the server comes back almost instantaneously. Which of the following is true ?

- [ ] foo completes its execution until its end, and goo is executed immediately afterwards
- [ ] goo is executed immediately when the response is returned from the server
- [ ] goo may not be executed until the message has been successfully checked for errors
- [x] goo may not be executed until the current thread of execution including foo is done

Question - 5
MCQ 5

SCORE: **1 points**

If an object was constructed using `Object.create` instead of using constructor functions, which of the following operators would not make sense to use with the object ?

- [x] instanceOf
- [ ] in
- [ ] typeof
- [ ] hasProperty

Question - 6
MCQ 6

SCORE: **1 points**

Which of the following statements in JavaScript can NOT be used to emulate calling a function `foo` on the object `obj` as though `foo` was a method of the object `obj` (i.e., equivalent of `obj.foo()`) ? Assume that `foo` does not take any arguments in this question.

- [ ] foo.apply(obj, [ ])
- [ ] foo.call(obj)
- [ ] foo.bind(obj)()
- [x] None of the above

Question - 7
Higher-Order Functions

SCORE: **5 points**

Javascript Functions

Write a function `wrapFunc` that takes three arguments, `preFunc`, `func`, and `postFunc`, each of which are themselves functions. The `wrapFunc` function should return a function that calls `preFunc` first, followed by `func`, and then finally `postFunc`. The function returned by `wrapFunc` should pass its arguments to `func`, as though `func` had been called by itself (you may not make any assumption about the types or number of arguments of `func`), and it should return the value that `func` would have returned. The functions `preFunc` and `postFunc` takes a single argument each, namely the `func` function.

Example Usage

```
function foo() {
    var argStr = "Called foo with arguments:\n";
    for (let arg of arguments) {
        argStr += arg.toString() + " ";
    }
    console.log(argStr);
    return 'End';
}

function preLog(f) {
    console.log("Before Calling " + f.name);
}

function proLog(f) {
    console.log("Done calling " + f.name);
}

function wrapFunc(func, preLog, proLog) {
    // Enter code here ...
}
```

Restrictions

- You may not add any global variables or else no points will be given.

Input:

```
var g = wrapFunc(foo, preLog, proLog);
console.log( g(1, 2, 3) );
```

Output:

```
Before Calling foo
Called foo with arguments:
1 2 3
Done calling foo
End
```

Question - 8

DOM Traversal

SCORE: **5 points**

In the "JavaScript" tab, complete the implementation of the `findString(id, text, period, onFound)` function.

When the `findString` function is invoked, it should traverse the DOM subtree rooted at the node with the given `id`, and look for the given `text` in one or more text nodes in the subtree. Note that the `text` can be a substring of another string rooted at that node.

- If the `text` is found, invoke the `onFound` callback function.
- If the `text` is not found, then it must attempt to find the `string` again after the given `period`, specified in milliseconds. It must keep attempting to find the `string` periodically until the `text` is found. When it is found, invoke the `onFound` callback function as mentioned above.
- Arguments:

- `id` is a string; it is the `id` of the DOM element under which to search the given `text`
- `text` is a string; it is the substring to search for
- `period` is a number; it is the time interval (in millisecond) to wait for before performing another search
- `onFound` is a function; it should be invoked to indicate that the given `text` was found

Example Usage:

For example, given the DOM

```
<div id="example">
  Hello <span>world</span>!
</div>
```

and assuming `<p>Foobar</p>` is appended to the `div#example` at t=2500 ms, when `findString("example", "Foo", 1000, function(){ console.log("Found Foo!") })` is called at t=0ms,
it should find nothing first, and it should traverse the DOM again at t=1000ms, t=2000ms, and finally t=3000ms. When it traverses the DOM at t=3000ms, it should find the substring "Foo" in the newly appended `<p>` element, and then "Found Foo!" should be printed in the console.

Restrictions:

- You may not add any global variables to the program, or else no marks will be given. You may also not use any external libraries or frameworks for this question.
- You can only write code in the "JavaScript" tab.

How to use the tests:

There are 9 sub-tests that you can run. Each sub-test will invoke the `findString` function, passing in certain arguments. Click the "Run Test" button to invoke the function on the test node. You must click "Reset" before you run the test again (otherwise the test will fail even if your implementation is correct).

Question - 9

SCORE: **5 points**

Objects

Javascript | Object Oriented Programming

Note: For this question, you may receive partial points for passing tests for only part A (2 points) or only part B (3 points).

A. In this question, you will need to write a function *`inherits`* that takes as arguments a constructor function `foo`, and an object `obj`, and makes al
Further, whenever the constructor property of the `foo.prototype` is queried, it should point to `foo`.
(2 Points)

You may not use ES6 syntax for this question, nor can you make any assumption about the constructor function `foo`.

B. Consider the hierarchy of types below, representing different shapes. Let's assume `Point` is the parent type, and `Circle` is a child type.
How will you use the above *`inherits`* function to create a new `Circle` object in two steps similar to how we instantiated `Circle` objects with co
Note that the `Circle` object created must be almost the same as how we created the object using the `Circle` constructor functions in *shapes.js*.
You should use it to implement the function `createCircle(x, y, r)` which is expected to return an object with properties including x, y, and r.
The returned object should be the same as instantiating a circle via prototypical inheritance.
(3 Points)

```
var Point = function (x, y) {
   this.x = x;
   this.y = y;
   this.area = function() {
       return 0;
   }
};

Point.prototype.toString = function() {
   return "(" + this.x + "," + this.y + ")";
};
```

```
var Circle = function(x, y, r) {
   Point.call(this, x, y);
   this.r = r;
   this.area = function() {
       return 3.1412 * this.r * this.r;
   }
};
```

Question - 10

Node.Js

SCORE: **5 points**

Node.js

In this question, you will write a node.js program to open a text file in `streaming` mode, and read its inputs. Your goal is to recognize all the words within parentheses in the text file, and ***print their count.*** The program should work even if the words within the parentheses are split across blocks. A ***word*** is defined as a **contiguous**, **non-empty stream of non-space characters** - words can be preceded by `"("` and succeeded by `")"` in lieu of spaces, though. E.g., `"(hello "` is a word, as is `"world) "`, and so is `" hello "`, as well as `"   hello   "` (i.e., ***number of spaces don't matter***)

Some **conditions** and caveats:

1. Any number of contiguous `"("` can be used before a stream of words,
2. Any number of contiguous `")"` can be used after a stream of words,
3. You may assume that the text has ***balanced*** "(" and ")", and is **well formed** (e.g., we don't have test cases like "((Hello)" or "(World)))").

Note: you cannot make any assumptions about the size of the blob that is read, and your program should operate in streaming mode, i.e., it should count the words as and when they are read and not wait till the end (no points will be given if either condition is violated).

HINT: Keep track of whether you're within parentheses by using a counter and incrementing and decrementing it appropriately.