Question - 1

MCQ 1.

SCORE: **1 points**

Javascript

Consider the HTML file shown below and the corresponding CSS rules (for simplicity, this is shown in the same file). The color of the element div with ID “two” is now “green”. Assume you want to change it to red without changing any of the CSS rules. Which of the following aspects of the element will you need to change for this to happen?

```
<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>
```

- ID of the element
- Class of the element
- Tag of the element
- None of the above

Question - 2

MCQ 2

SCORE: **1 points**

This question also pertains to the same code sample from the previous question. In the HTML file, assume you want the div element with ID “five” to be of color “green” (currently it is yellow), but you may not make any changes to the HTML file. Which of the following rules will you need to remove from the CSS file for this to happen (only one rule may be removed, and the color of all the other elements should not be changed)?

```
 <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>
```

- Rule 1
- Rule 2
- Rule 3
- Rule 4

Question - 3

MCQ 3

SCORE: **1 points**

Consider the function below that attempts to find the largest element in an array passed to it as an argument, and throws an exception if an element of the array is not a number. You need to modify it such that it returns the largest element upto that point. Which of the following lines of the code needs to be modified to reflect this (assume you can only change one of these lines)?

```
  function largest(arr) {
        if (arr.length==0) throw { msg: "empty
array" };       // Line 1
        var result = arr[0];
        for (var i=0; i<arr.length; i++) {
                if (! isNaN(arr[i])) {
                        result = (result <
arr[i]) ? arr[i] : result;    // Line 2
                } else {
                        throw { msg: "Not a
number", index: i };  // Line 3
                }
        }
        return result;          // Line 4
   }
```

- ○ Line 1
- ○ Line 2
- ◉ Line 3
- ○ Line 4

Question - 4

MCQ 4

SCORE: **1 points**

When you send an AJAX message to the server using `xhr.send()`, and you want to setup the handlers for timeouts, aborts and errors, which of the following is *`true`* for the code to be correct:

- ○ The handlers must be initialized before xhr.open()
- ○ The handlers must be initialized after xhr.open() but before xhr.send()
- ○ The handlers must be initialized after xhr.send() but before the end of the current context
- ◉ The handlers must be initialized any time before the end of the current context

Question - 5

MCQ 5

SCORE: **1 points**

If you want an event handler '`foo`' to be registered for all the sub-elements of a DOM node '`d`' such that '`foo`' is invoked before any other handler of `d` or any of its sub-elements is invoked, you will register it using which of the following ?

- Capture handler on node ‘d
- Bubble handler on node ‘d’
- Capture handler on the parent of ‘d’
- Bubble handler on the parent of ‘d’

Question - 6

MCQ 6

SCORE: **1 points**

Which of the following is `true` about the binding of `'this'` value in regular functions and arrow functions (ES6) ?

- ‘this’ is lexically bound in both regular and arrow functions
- ‘this’ is lexically bound in arrow functions, but dynamically bound in regular functions
- ‘this’ is dynamically bound in arrow functions, but lexically bound in regular functions
- ‘this’ is dynamically bound in both regular and arrow functions

Question - 7

Closures

SCORE: **5 points**

Javascript

Write a function `cacheArguments` that takes a function `foo` and a number `n` as its arguments. It should return a function that remembers the arguments of `foo` upto the number `n`, from the last time `foo` was invoked. In case, `foo` is invoked with fewer arguments than `n`, the arguments from the last invocation of `foo` upto `n` should be provided in place of the missing arguments (if there was no such previous invocation, you can pass undefined). Note that the arguments from the previous invocation should be used if and only if fewer than `n` arguments are specified in the invocation, and that no more than `n` arguments from the previous invocation should be used.

For this question, you may NOT

1. Add any global variables to the program
2. Make any assumptions about the number of arguments of `foo`, except that it’s greater than `n (n > 0)`.
3. Modify the behavior of `foo` in any way either in its invocation or return value other than the above

Note that violation of any of these constraints will result in you earning a 0 for this question.

HINT: Use closures to cache the arguments upto ‘n’ from the previous invocation of foo.

```
function foo{ return arguments;}
var f = cacheArguments(foo, 4);

f(1, 2, 3, 4); // return 1, 2, 3, 4
f(10, 20);  // return 10, 20, 3, 4
f(1);       // return 1, 20, 3, 4
f(1, 2, 3, 4, 5); // return 1, 2, 3, 4, 5
f(); // return 1, 2, 3, 4
```

Question - 8

DOM Traversal

SCORE: **5 points**

In the "JavaScript" tab, complete the implementation of the `watchTextChange(node, hasChanged)` function.

The `watchTextChange` function accepts 2 arguments: `node` and `hasChanged`.

- `node` is a DOM `Node` object.
- `hasChanged` is a function that accepts a single argument `node`, which is a `Node`, and returns `true` if the `node` has changed since its last invocation, and returns `false` otherwise. *(You don't need to implement this function, it's given to you.)*

1. Given a `node`, the `watchTextChange` function should traverse the `node` and all its subtrees and **count the number of text nodes (nodeType = 3) whose text has changed** - you can use the `hasChanged` function to determine whether the text has changed or not. The function should return the count.
2. In case a given text node has changed, **add the CSS class `changed` to its parent node**. If a node has the CSS class `skip`, **do not count the text nodes under it** even if they change values.

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 `watchTextChange` function, passing in a particular `node` object. 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).

- When your function returns the correct count, the test will output "Count OK"
- An element with the CSS class `changed` will have a green background. An element with the CSS class `skip` will have a gray background.

Question - 9

Objects

SCORE: **5 points**

Javascript | Object Oriented Programming

A.

In this question, you will write your own version of the *new* operator *newOp* without using *new* that takes the following arguments: *Obj (*prototype object), *foo* (constructor function), and *args (*array of arguments to pass to the constructor function). You need to emulate the behavior of the *new* operator, but with one important difference: the new object should not use the foo.prototype as its prototype object, but rather the Obj object as the prototype. In other words, even though the constructor function is called, it does not instantiate an object of its type as it normally would. (3 Points)

You may not use ES6 syntax for this question, nor can you make any assumption about the number of arguments taken by the constructor function. Further, you should mirror the exact behavior of the *new* operator with the above exception, or else no points will be given.

```
function newOp(obj, foo, args) {

}
```

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 newOp function to create a new circle object in two steps similar to how we instantiated Circle objects with constructor functions in class ? You cannot use the new operator or Object.create directly, but you must use the above *newOp* function exactly two times, or else no points will be given. 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. This time you need 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 when we instantiated a circle doing prototypical inheritance, unless it is not considered as an instance of Circle but it is considered as an instance of Point. (lecture 2). (2 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 async input handling

SCORE: **5 points**

Node.js

In this question, you will write a Node.js program that uses `ReadStream` to read a blob of text in streaming mode. Your goal is to recognize star words in the text, and count their occurrences. When the program ends (after the stream is read), it should print the count of star words in the file - ***a single integer value in String format***.

A star word is defined as a contiguous, non-empty stream of non-space characters that is preceded by two or more '*' characters, and succeeded by one or more spaces (newlines do not count as spaces).
Note that there may be multiple stars before a word, and multiple spaces after a word – these should not be identified as separate star words. For example, `"***example   "` is a single star word.
On the other hand, `"example "` is not a star word as it does not have a sequence of `"*"` preceding it.
Similarly, `"**** "` by itself is not a star word as it is empty (there's a space right after the last *).
Also, you shouldn't count *s already within a word like `"*example**s "` as star words either, since the double `"*"` does not start a new word.
If the total number of star words exceeds 10, print `-1`.

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