Section-A (60 Marks)

Question 1-5 contains 1 mark each:

1. Which of the following is the correct way to create a list using the lowercase letters?

a. <ol alpha=”a”>

b. <ol type=”a”>

c. <ol letter=”a”>

d. None

Answer. **b. <ol type=”a”>**

2. In HTML5, which of the following tag is used to initialize the document type?

a. <Doctype HTML>

b. <\Doctype html>

c. <DOCTYPE>

d. <!DOCTYPE html>

Answer. **d. <!DOCTYPE html>**

3. Which of the following has the highest priority in CSS?

a. #id

b. HTML tags

c. .my-class

d. :hover

Answer. **a. #id**

4. The default value of “position” property is _____.

a. fixed

b. absolute

c. static

d. relative

Answer. **c. static**

5. How to stop event bubbling in JavaScript?

a. e.stopBubble()

b. e.stop()

c. e.pauseBubble()

d. e.stopPropagation()

Answer. **d. e.stopPropagation()**

6. Explain Closure, Hoisting and Currying (Function Composition) in JavaScript, with the help of examples. **(10 Marks)**

Answer.

A closure may be described as a combination of a function and the lexical environment in which it was declared. The lexical environment consists of any local variables in the function's scope when the function is created. A closure enables one to refer to all local variables of a function in the state they were found.

This is essentially achieved by defining a function inside another function, this function within a function is technically the closure. Each time the parent function is called, a new context of execution is created holding a fresh copy of all local variables. These local variables can be referred to in the global scope by either linking them to variables declared globally or returning the closure from the parent function.

A basic example will take on a format similar to this:

```
function closuredFunc (){
   function closure(){
   // some logic
   }
}
```

It is also possible to have a closure that returns several methods as shown below:

```
 function closure(){
   function first() { console.log('I was declared first')}
   function second() { console.log('I was declared second')}
```

```
    function third() { console.log('I was declared third')}

    return [first, second, third]

}
```

To reference each of these methods, we'll assign our closure to a global variable which will then point to an array of exposed methods. Each method can then be assigned to unique variable names to bring them into the global scope as shown below. At this point, they can now be called.

```
let f = closure()

let one = f[0]

let two = f[1]

let three = f[2]


one() // logs I was declared first

two() // logs I was declared second

three() // logs I was declared third
```

Why Use Closures?

You may be wondering why would one go through the trouble of making closures. Well, closures have a number of uses and advantages. Prior to the introduction of Classes in ES6, closures provided a means of creating class-like privacy similar to that used in Object Oriented Programming, allowing us to emulate private methods. This is known as the module pattern and it allows us to write easily maintainable code with reduced namespace pollution and more reusability.

What is Currying?

Currying is the pattern of functions that immediately evaluate and return other functions. This is made possible by the fact that Javascript functions are expressions that can return other functions.

Curried functions are constructed by chaining closures by defining and immediately returning their inner functions simultaneously.

Here's an example of currying:

```
let greeting = function (a) {
   return function (b) {
      return a + ' ' + b
   }
}
let hello = greeting('Hello')
let morning = greeting ('Good morning')

hello('Austin') // returns Hello Austin
hello('Roy') // returns Hello Roy
morning('Austin') // returns Good morning Austin
morning('Roy') //returns Good Morning Roy
```

The two functions created from greeting (hello and morning) each return functions that process the provided inputs to generate a greeting statement. They also take an argument which is the name of the person to be greeted.

In the above case, greeting is also used as a function factory with the two functions hello and morning generated from it.

The inner function may also be called invoked after the first call as follows:

```
greeting ('Hello There') ('General Kenobi')

//returns Hello There General Kenobi
```

Currying is considered to be part of functional programming and as such curried functions may be easily written using the arrow function syntax in ES6 and newer versions of Javascript for cleaner, more elegant code:

```
let greeting = (a) => (b) => a + ' ' + b

greeting('Hello There')('General Kenobi')

//returns Hello There General Kenobi
```

JavaScript Declarations are Hoisted. In JavaScript, a variable can be declared after it has been used. In other words; a variable can be used before it has been declared.

Example:

```
x = 5; // Assign 5 to x

elem = document.getElementById("demo"); // Find an element

elem.innerHTML = x;                    // Display x in the element


var x; // Declare x
```

7. Demonstrate using code sample, multiple ways of using CSS in HTML. What is the priority order of the same? **(10 Marks)**

ANSWER.

CSS gives flexibility in using different style property locally by overriding the global declared values or declared styles in external style sheet.

Let us start with Types of styles. Mainly there are three types:

1. Inline style sheet Or embedded styles

2. Internal style sheet

3. External style sheet

We also have another two types of styles at the client end. One is user defined style and other is browser default style. If more than one style is defined then which one will be followed? For this there is a priority level defined and based on this priority the styles property gets included.

User defined style Embedded or inline style sheet Internal style sheet

External style sheet Browser default style

We can see here that out of the above list as a designer we have control over serial number 2,3 and 4. These three types of styles we will discuss more. Within these three (Inline, Internal & External) the priority order is first In line, then Internal and last priority is given to external styles. This is the biggest advantage as we can override the global style property and define them locally. Let us start learning each of these three types of CSS.

Inline style:

We add the styles within our HTML tags. This gets highest priority than Internal and external defined styles. Let us try to add a background style to one h4 tag.

```
<h4 style="background-color: #f1f1f1;">This is H4 tag</h4>
```

Internal Style:

This type of style is defined inside the body tag of the page. The style defined here get priority over the external styles but after the Inline styles. Here is an example of Internal styles.

```
<html>
<head>
<title>(Type a title for your page here)</title>
<style type="text/css">
<!--
p {
 font-family: verdana, arial, sans-serif;
background-color: #ffff00;
 }
-->
</style>
</head>
<body >
<p> This is content and its style is defined
within the internal style sheet</p>
</body>
</html>
```

External Style sheets

These styles are kept separately and called or linked from the required page. This is the syntax followed for linking a external style sheet.

<link rel="stylesheet" href="images/style.css" type="text/css">

The above tag is to be placed within the head tags of the page. We can keep our style details inside the file named style.css. Watch the linking of the file. The path details are to be given for including the style sheet.

8. Find and explain the output of the following code snippet**.**

(5 marks)

```
const arr = [10, 12, 15, 21];
for (var i = 0; i < arr.length; i++) {
  setTimeout(function() {
    console.log('Index: ' + i + ', element: ' + arr[i]);
  }, 3000);
}
```

ANSWER. The output will be as follows: -

1.At Index "0" it will be element 10

2. At Index "1" it will be element 12

3. At Index "2" it will be element 15

4. And at Index "3" with element 21

As we have used setTimeout() method which will Increase the output time of all the console outputs an enormous increase of 3000milliseconds that is 3 seconds.

The setTimeout() method calls a function or evaluates an expression after a specified number of milliseconds.

1000 ms = 1 second.

The function is only executed once. If we need to repeat execution, we have to use the setInterval() method.

We can use the clearTimeout() method to prevent the function from running.

9. Find and explain the output of the following code snippet.

(5 marks)

```
const promise1 = new Promise((resolve, reject) => {
  console.log(1);
  resolve('success')
});
promise1.then(() => {
  console.log(3);
});
console.log(4);
```

ANSWER. Output will be as follows:

1 4 3

Well at first stage as new promise is on the run it will simply console log 1 and then wait for the successful implementation, after the completion of the above it will wait for the response to send back, in the meantime it will console log "4" and as soon as response is recorded at server it will simply exit the code on console.