Date: 02-6-2020
Morning Session: 9 am – 11 PM
By ~ Sundeep Charan Ramkumar Today

Topics: Scope and closures

Scope: is nothing but a confined area where you can define variables as well as your functions.

Functional Scope: Whenever you declare a variable within a function, that variable is *only available within that function*.

```
function sayHi() {
    var greeting = "Good morning";
    console.log(greeting);
}
sayHi(); //result = Good Morning
greeting(); // ReferenceError: greeting is not defined
```

Global Scope: If you declare a variable outside of any function or curly brackets { } then it is in the global scope.

```
var greeting = "Hello world!"; // global scope

function myFunc() {
  console.log(greeting);
}

myFunc(); // Result = "Hello world"
```

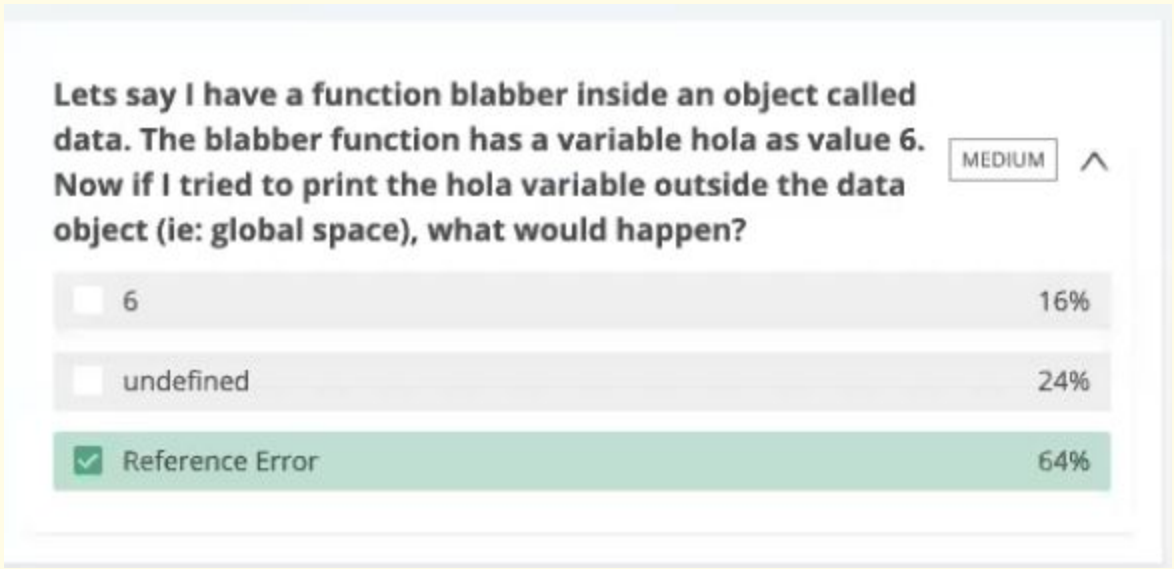
Window Global object : The global object provides variables and functions that are available anywhere. By default, those that are built into the environment. In a browser it is named window.

```
var message = "Hi there";
function outerFunction() {
  message = "Hello there";
  function innerFunction() {
    message = "Bye there";
    console.log("Inner function called ", message);
  }
  innerFunction();
  console.log("Outer function called ", message);
}

console.log("Window object scope ", message);
outerFunction();
console.log("Still outside the window object ", message);
```

Note: scope chain works from top bottom, does not work from bottom to top

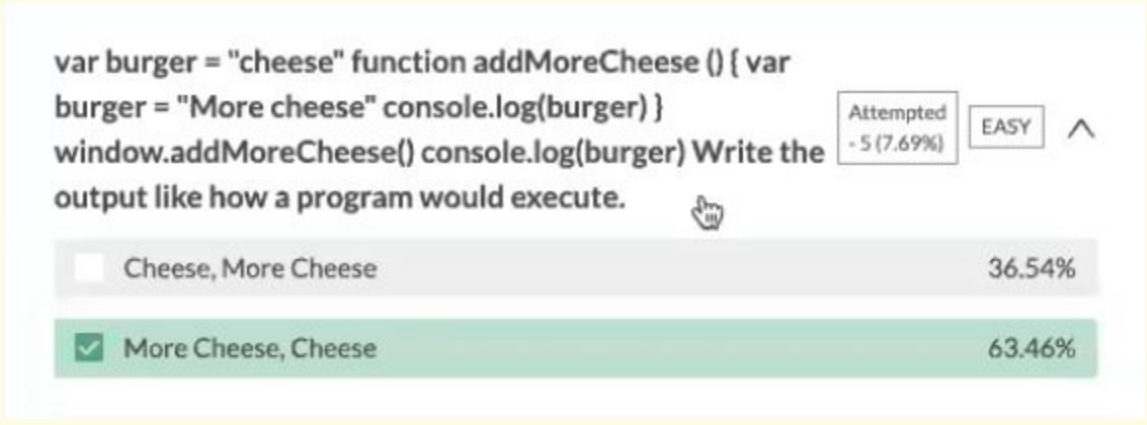
MCQ NO: 1



```
var data = {
  blabber: function () {
    var hola = 6;
  }
};

console.log(hola);
```

MCQ 2:




```
var burger = "cheese";
function addMoreCheese() {
  var burger = "More cheese";
  console.log(burger);
}
window.addMoreCheese();
console.log(burger);
```

Closures: when a function is getting called its memory resources will get freed from the computer so that we cannot access the variable but there are exceptions. In closures the variable will still be stored inside the memory is called closure.

It has 2 points.

1) Return the function

2) Variable stored & not get deleted even after invoke

Currying Function?

A curried function is a function that takes multiple arguments *one at a time.*

Given a function with 3 parameters, the curried version will take one argument and return a function that takes the next argument, which returns a function that takes the third argument. The last function returns the result of applying the function to all of its arguments.

```
function a() {
    return function b() {
      return function c() {
        return "I am the third level function";
      };
    };
  }

  var bFunction = a();
  var cFunction = bFunction();
  var cFunctionReturnText = cFunction();
  console.log(cFunctionReturnText);

  console.log(a()()());
```

```
function makeMultiplicationTable(firstNumber) {
    var pizza = "mozarella";
    return function (secondNumber) {
      console.log(pizza);
      return function (thirdNumber) {
        return firstNumber * secondNumber + thirdNumber;
      };
    };
  }

  var sixTable = makeMultiplicationTable(6);
  var sevenTable = makeMultiplicationTable(7);
  console.log(sixTable(9));
  console.log(sevenTable(10));
```

It stores the value of the memory unless the inner function gets called.

MCQ 3:

MCQ4:

Resources:

Arrays and Objects:
https://www.youtube.com/watch?v=FLGzeTHAbqQ

https://www.youtube.com/watch?v=WOBlhuGHbdE

Functions:

https://www.youtube.com/watch?v=AY6X5jZZ_JE

https://www.youtube.com/watch?v=xUI5Tsl2JpY

Prototype and Prototypical inheritance:
https://www.youtube.com/watch?v=YkoelSTUy7A

https://www.youtube.com/watch?v=7oNWNlMrkpc

https://www.youtube.com/watch?v=ullj6_z_wL8

Closures:
https://www.youtube.com/watch?v=71AtaJpJHw0

https://www.youtube.com/watch?v=3a0I8ICR1Vg