Ques 1: What is the difference between Cookie / Local Storage / Session Storage?

Ans: **Cookies:** Cookies are data stored in small text files of computer. Cookies are used to store some information about the user when user visits a link. This information can be used for the future reference. The storing size of a cookie is 4kb and you cannot store more than that data.

Auto Expire option is present.

Supports SSl.

Can be accessed from serve side.

**Session:** The session storage properties allow saving key value pair in a web browser. The sessions stored are only limited up to the browser or window is open as the browser closes the session storage is cleared. The storing size in a session is 5mb and you store as much data as of 5mb.

Auto Expire option is present.

No SSl support.

No Accessible server side.

**Local Storage:** The local Storage object stores the data with no expiration date. The data is not deleted when the browser is closed, and will be available next day, week or year.

Auto Expire option is not present

No ssl support.

No Accessible server side.

Ques: 2 What is the significance of, and reason for, wrapping the entire content of a JavaScript source file in a function block?

* The main reason of wrapping content of js source file in a function block is to run an error free script.
* The variables declared in a scope are private to that scope.
* It doesn’t affect the global scope.
* Global variables are generally considered bad and cause more errors in code.
* On running a code the browser compiles all js files as one so declaring global variables can cause overriding of code if same variable is declared somewhere else.

Q3. What will the code below output? Explain your answer

console.log(0.1 + 0.2); = 0.30000000000000004;

console.log(0.1 + 0.2 == 0.3); //

It will give false because of floating point math. The computer uses binary floating point format and human follow decimal point format.

Q4. Write a sum method which will work properly when invoked using either syntax below.

function addNumber(fn, totalArgs){

//console.log(fn.length,arguments.length);

totalArgs = totalArgs || fn.length

return function recursor(){

return arguments.length < totalArgs ? recursor.bind(this, ...arguments): fn.call(this, ...arguments);

}

}

var add = addNumber((a,b)=> a+b);

console.log(add(2)(3));

console.log(add(2)(3));

Q5. What is the output of the following code?

var length = 10;

function fn() {

console.log(this.length);

}

var obj = {

length: 5,

method: function(fn) {

fn();

arguments[0]();

}

};

Output = 2;

In the first place, as fn is passed as a parameter to the function method, the scope (this) of the function fn is window.

var length = 10; is declared at the window level. It also can be accessed as window.length or length or

this.length (when this === window.)

method is bound to Object obj, and obj.method is called with parameters fn and 1.

Though method is accepting only one parameter, while invoking it has passed two parameters;

the first is a function callback and other is just a number.

When fn() is called inside method, which was passed the function as a parameter at the global level,

this.length will have access to var length = 10 (declared globally) not length = 5 as defined in Object obj.

Now, we know that we can access any number of arguments in a JavaScript function using the arguments[] array.

Hence arguments[0]() is nothing but calling fn(). Inside fn now, the scope of this function becomes the arguments array,

and logging the length of arguments[] will return 2.