**Question No: 1**

|  |  |  |
| --- | --- | --- |
| Global Context  Phase 1 :   * Var x = undefined * Var a = undefined * Var b = undefined * Var c = undefined   Phase 2: Executing   * Var x = 1; * Var a = 5 * Var b = 10 * Var c = function(..) * **C(8,9,10)** * Print b => 10 * Print x => 1 | C Context  Phase 1:   * Var a = undefined * Var b = undefined * Var c = undefined * Var f = undefined * Var x = undefined   Phase 2:   * Var a = 8 * Var b = 9 * Var c = 10 * Print x => undefined * Print a => 8 * Var f = function(…) * **F(a,b,c)** * Print b => 9 * Var x = 10; | F Context  Phase 1:   * Var a = undefined * Var b = undefined * Var c = undefined * Var x = undefined   Phase 2:   * Var a = 8 * Var b = 9 * Var c = 10 * (b = a) => (b = 8) * Print b => 8 * (b=c) => (b=10) * Var x = 5 |

From the above table, the result will be look like: Undefined 8 8 9 10 1

**Question No: 2**

On java there are only two types of scope before ES6 , Global scope and local scope.

* **Global Scope:** when there is a variables which is visible by every function, then it will be called global scope.
* **Local Scope:** when a variable is visible to a specific function and its descendants, then it will be called local scope.

**Question No: 3**

A: No

B: Yes

C: No

D: Yes

E: Yes

Reason: The reason for the above answer is: It is possible to access from higher scope to lower scope. But it is not possible to access lower scope to higher scope level.

**Question No: 4**

|  |  |
| --- | --- |
| Global Context  **Phase 1 : hoisting**   * Var x = undefined * Function(…)   **Phase 2 : Executing**   * Var x = 9 * Function(…) * **Document.write(myFunction());** * Var x = 5 * **Document.write(myFunction());** | myFunction Context  **phase 1 :**   * var x = undefined;   **phase 2 :**   * print x \* x => 9 \* 9 = 81   myfunction context  **Phase 1:**   * Var x = undefined;   **Phase 2:**   * Print x\*x => 5 \* 5 = 25 |

Based on the above table: the result is: **81 25**

**Question No: 5**

|  |  |
| --- | --- |
| Global Context  **Phase 1 : hoisting**   * Var foo = undefined * Function(…)   **Phase 2 : Executing**   * Var foo = 1 * **Bar();** | bar Context  **phase 1 :**   * var foo : undefined;   **phase 2 :**   * !foo = true; * Var foo = 10; * Alert (foo) => alert 10. |

Based on the above table: 10 will be alerted.

**Question 6:**

1. Const count = {
2. Var counter = 0;
3. Add: **function**() {
4. Counter = counter + 1;
5. }
6. Reset:**function**(){
7. Counter = 0;
8. }
9. }

**Question 7:**

Counter is free variable here.

**Question 8:**

1. Const count = {
2. Var counter = 0;
3. Add: **function**() {
4. Counter = counter + 1;
5. }
6. Reset:**function**(){
7. Counter = 0;
8. }
9. Make\_adder:function(inc){
10. Counter = counter + inc;
11. }
12. }

**Question 9:**

* The solution is to use Immediately Invoked Function Expression (IIFE)
* Or if we are using ES6 , just add curly braces can fix the problem.

**Question 10:**

1. **const** Module = (**function**() {
2. let name;
3. let age;
4. let salary;
6. **const** getAge = **function**() {
7. **return** age;
8. };
9. **const** getSalary = **function**() {
10. **return** salary;
11. };
12. **const** getName = **function**() {
13. **return** name;
14. };
15. **const** increaseSalary = **function**(percentage) {
16. salary = getSalary() \* (percentage / 100);
17. };
18. **const** incrementAge = **function**() {
19. age = age + 1;
20. };
21. **const** setAge = **function**(newAge) {
22. age = newAge;
23. };
24. **const** setSalary = **function**(newSalary) {
25. salary = newSalary
26. };
27. **const** setName = **function**(newName) {
28. name= newName;
29. };
30. **return** {
31. setAge: setAge,
32. setSalary: setSalary ,
33. setName: setName,
34. increaseSalary: increaseSalary,
35. incrementAge: incrementAge
36. };
37. })();

**Question 11:**

1. Module.address;
2. Module.getAddress = **function** () {
   1. **return** address;
3. }
4. Module.setAddress = **function** (newAddress) {
   1. Address = newAddress;
5. }