JAVA SCRIPT

1) What will be the output of this code?

```
console.log(x);
var x=5;
```

o/p: undefined

Explanation:

Step 1: The declaration of var x is hoisted to the top of the scope, but the initialization (x = 5) is not.

Step 2: When console.log(x) is executed, x has been declared but not yet assigned a value, so it's undefined at that point.

After the console.log, x is assigned the value 5, but this happens after the log statement.

2) What will be the output of this code?

```
console.log(x);
var x;
```

o/p: undefined

Explanation:

Step 1: The variable x is declared at the top of the scope due to hoisting.

Step 2: console.log(x) is executed, the variable x exists, but it hasn't been assigned a value, so it logs undefined.

There is no assignment after the declaration, so x remains undefined throughout.

3) What will be the output of this code?

```
console.log(a);
a=10;
var a;
```

o/p: undefined

Explanation:

Step 1: The variable a is declared (hoisted) at the top of the scope, but it's not yet assigned any value, so it's undefined at the time of console.log(a).

Step 2: console.log(a) is executed, and since a is declared but not initialized, the output is undefined.

Step 3: After logging, a is assigned the value 10, but this happens after the log statement.
Hence, the output will be undefined.

4) What will be the output of this code?

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

o/p: Reference Error

Explanation:

Step 1: In this case, a has not been declared anywhere in the code before console.log(a) is executed.
So you will get a ReferenceError

5) What will be the output of this code?

```
console.log(a);
  var a=10;
  console.log(a);
  a=20;
  console.log(a);
```

o/p: undefined
10
20

Explanation:

Step 1: The variable a is declared at the top because var declarations are hoisted. Before the assignment of 10, a is undefined.

Step 2: (console.log(a)): At this point, a is still undefined because it hasn't been assigned any value.

Step 3: (a = 10): The variable a is assigned the value 10.

Step 4: (console.log(a)): Now, a holds the value 10, so 10 is logged.

Step 5: (a = 20): The variable a is reassigned the value 20.

Step 6: (console.log(a)): Now, a holds the value 20, so 20 is logged.

6) What will be the output of this code?

```
console.log(f);
  var f=100;
  var f;
```

```
console.log(f);
```

o/p: undefined
100

Explanation:

Step 1: When var f; is hoisted to the top, f is created with an initial value of undefined.

Step 2: console.log(f): At this point, f is still undefined since it hasn't been assigned any value yet. Thus, it outputs undefined.

Step 3: (f = 100): Here, f is assigned the value 100.

Step 4: (var f): This line does not change anything because f has already been declared. The var keyword allows redeclaration but does not affect the existing value.

Step 5: console.log(f): Now, since f has been assigned the value 100, this logs 100.

7) What will be the output of this code?

```
console.log(g);
 var g=g+1;
console.log(g);
```

o/p: undefined
NaN

Explanation:

Step 1: console.log(g): output will be undefined because the declaration of g is moved to the top, but not its initialization with the help of hosting.

Step 2: (**var g = g + 1)**: attempts to add 1 to g, but g is undefined.

Step 3: console.log(g): output is NaN because value is not assigned for g.

8) What will be the output of this code?

```
 var h;
console.log(h);
h=50;
console.log(h);
```

o/p: undefined
50

Explanation:

Step 1: The line var h; declares the variable h. In JavaScript, when a variable is declared but not initialized, it is automatically assigned the value undefined.
Step 2: console.log(h): When this line is executed, h has been declared but not assigned a value yet. Therefore, it outputs undefined.
Step 3: Assignment (h = 50): Here, h is assigned the value 50.
Step 4: console.log(h): Now that h has been assigned the value 50, this log statement outputs 50.

9) What will be the output of this code?

```
console.log(i);
i=10;
var i=40;
console.log(i);
```

o/p: undefined
40

Explanation:

Step 1: The line var i; is hoisted, so i is declared but not initialized. Therefore, i is undefined at this point.
Step 2: console.log(i): Since i has not yet been assigned any value, it outputs undefined.
Step 3: (i = 10): Now, i is assigned the value 10, but this happens after the first log statement.
Step 4: (var i = 40): This line re-declares i, but since i is already declared, it just updates the value of i to 40.
Step 5: console.log(i): Finally, this logs the current value of i, which is now 40.