Task

1) **What will be the output of this code?**

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

output= undefined

Explanation; we know about interpreter that it will execute code line by line. Firstly it will execute JavaScript engine then scan the code and stores the memory for each variable but it considers default value as undefined rather than user defined value that's why the output will be undefined.

2) **What will be the output of this code?**

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

output= undefined

Explanation; In JavaScript, variable declarations with `var` are hoisted, meaning `var x;` is processed before any code runs. Although `x` is declared, it isn't initialized, so its value is `undefined` at the time of the `console.log(x);` call. Thus, it prints `undefined`.

3) **What will be the output of this code?**

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

output= undefined

Explanation; In JavaScript, variable declarations using var are hoisted to the top of their scope, so var a; is processed before any code runs. When console.log(a); is executed, a is declared but not yet initialized, resulting in a value of undefined. The assignment a = 10; occurs after the console.log, so it has no effect on the output at that point

4) **What will be the output of this code?**

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

output= Reference Error: a is not defined

Explanation: This occurs because the variable a has not been declared or initialized before the console.log(a); statement is executed. In JavaScript, trying to access a variable that hasn't been defined leads to this error

5) **What will be the output of this code?**

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

output= undefined 10 20

Explanation: because a is hoisted but not initialized yet, as var a = 10; hasn't executed. On line 4, console.log(a); outputs 10 since the assignment has now taken place. Finally, after line 5, a is reassigned to 20, so line 6 outputs 20. The sequence illustrates hoisting and how variable assignments affect logged values.

6) **What will be the output of this code?**

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

output= undefined 100

Explanation: because the variable f is hoisted but not initialized yet, as var f = 100; has not been executed. By line 11, f is assigned the value of 100. Therefore, when line 13 executes console.log (f);, it outputs 100, reflecting the updated value after the assignment. This demonstrates variable hoisting and initialization behaviour in JavaScript.

7) What will be the output of this code?

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

output= undefined NaN

Explanation because the variable g is hoisted but not initialized at that point. When the code reaches line 18, the expression g = g + 1; attempts to evaluate g, which is still undefined, leading to undefined + 1, resulting in NaN. Thus, when line 19 executes console.log(g);, it outputs NaN, indicating that the operation was invalid due to the uninitialized value.

8) What will be the output of this code?

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

output: undefined 50

Explanation: At first variable is declared but value is not assigned to variable and hence when we give console.log(h); it will give the default value of the variable that is undefined and then the value is assigned to the variable h here if we give console.log(h) then the output will be the value 50.

9) **What will be the output of this code?**

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

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

output=undefined 40

Explanation: from console.log(g)=undefined output will be displayed because only declaration i I will be moved to top of the code not its initialization bcoz of hosting. From i= 10 is assigned to the variable and from var i=40 reassigning for the variable is been done with the value of given value and finally from console.log(i)= output will be displayed=40