Variables in JavaScript

1. What will be the output of this code?

console.log(x);

var x=5;

Output: Undefined

Explanation: In JS when we declare a variable using var, that declaration is hoisted to the top of its scope, the declaration may be before or after the console.log(). In the above code we see that the console.log() is executed first and the variable is declared after the console.log() so the variable declared will be hoisted at the top of their scope. The events will be considered as declaration first “var x;” and the ”console.log(x);” next so the output is ”undefined”.

2. What will be output of this code?

console.log(a);

var a;

Output: Undefined

Explanation: In JS when we declare a variable using var, that declaration is hoisted to the top of its scope, the declaration may be before or after the console.log(). In the above code we see that the console.log() is executed first and the variable is declared after the console.log() so the variable declared will be hoisted at the top of their scope. The events will be considered as declaration first “var a;” and the “console.log(a);” next so the output is”undefined”

3. What will be the output of this code?

console.log(b);

b=10;

var b;

Output: Undefined

Explanation: The variable b is declared using var. This means that the declaration is hoisted to the top of the scope.so when JS Engine compiles the code, it hoists the declaration of b so the events will be considered as declaration first “var b;” and the “console.log(b);” next and it executes as the variable is declared but value is not assigned so the output is “undefined”.

4. What will happen here?

console.log(c);

Output: Reference Error: c is not defined

Explanation: Reference Error: c is not defined this is because until variables are declared with var, let, const. which are hoisted and initialised to undefined but variables declared with let, const will be hoisted but not initialised. This executing them before their declaration results in “Reference Error” When the JS engine executing console.log(c);, it looks for the declaration of c. Since there is no declaration of c anywhere in the scope and it wasn’t been defined with var, let, or const, the engine gives Reference Error and give the statement as “c is not defined”

5. What will be output of this code?

console.log(e);

var e=10;

console.log(e);

e=20;

console.log(e);

Output: Undefined

10

20

Explanation:

- Here the **console.log(e)** given above the e declaration and the variable e is declared with **var**, so the declaration is hoisted to the top of the scope. so it hasn't been assigned any value to the declaration , so the output is **undefined** here.
- Now in the next line value is assigned to **e** and the output for **console.log(e);** here is **10**.
- Now the value is updated here as **e=20**; so the output for **console.log(e);** here is **20**.

6. What will be the output of this code?

console.log(f);

var f=100;

var f;

console.log(f);

Output: Undefined

100

Explanation: The first `console.log(f)` outputs `undefined` because `f` is hoisted but not initialized before the log statement. The assignment `var = f = 100;` is a syntax error, so `f` remains uninitialized. The second `console.log(f)` would not execute due to the error, resulting in no output.

7. What will be the output of this code?

console.log(g);

var g=g+1;

console.log(g);

Output: Undefined

NaN

Explanation: The first `console.log(g)` outputs `undefined` because `g` is hoisted but not initialized when the log runs. The line `var g = g + 1; attempts to reference `g` before it has a value, leading to `g` being treated as `undefined`. Thus, `g` becomes `undefined + 1`, which results in `NaN` (Not a Number). The second `console.log(g)` then outputs `NaN`.

8. What will be the output of this code?

```
var h;

console.log(h);

h=50;

console.log(h);
```

Output: Undefined

50

Explanation: The first `console.log(h)` outputs `undefined` because the variable `h` is declared but not initialized. At this point, it exists in memory but has no assigned value yet. When `h` is assigned `50` in the next line, it now holds a defined value. The second `console.log(h)` then outputs `50`, reflecting the assignment. This illustrates variable declaration and initialization in JavaScript.

9. What will be the output of this code?

```
console.log(i);

i=10;

var i=5;

console.log(i);
```

Output: Undefined

5

Explanation: In JavaScript, variable declarations using `var` are hoisted to the top of their scope, but their assignments are not.

1. The first `console.log(i); outputs `undefined` because `i` is declared but not yet assigned a value.

2. Next, `i` is assigned the value `10`, but this doesn't affect the hoisted declaration.

3. The line `var i = 5; assigns `5` to `i`, but it's hoisted to the top of the function or global scope.

4. The second `console.log(i); outputs `5`, reflecting the most recent assignment.