Variables in js

1) What will be the output of this code?

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

Output: undefined

Explanation: -

This happens because of variable hoisting in JavaScript. When the code is executed, the declaration var x; is hoisted to the top, but the assignment x = 5; remains in its original place. So, when console.log(x); is executed, x is declared but not yet assigned a value, resulting in undefined.

2) What will be the output of this code?

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

Output: undefined

Explanation: -

This is due to variable hoisting in JavaScript. The declaration var a; is hoisted to the top, but the assignment doesn't happen until later, so when console.log(a); is executed, a exists but hasn't been assigned a value yet, resulting in undefined.

3) What will be the output of this code?

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

Output: undefined

Explanation: -

Here's why: The declaration var b; is hoisted to the top of the scope, meaning that at the time of the console.log(b); statement, b is declared but not yet assigned a value. Since b hasn't been initialised yet, it outputs undefined. The assignment b = 10; happens after the console.log call.

4) What will happen here?

```
console.log(c);
```

Output: Reference Error: c is not defined

Explanation: -

Since variable `c` was never declared, JavaScript will throw a ReferenceError.

5) What will be the output of this code?

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

```
console.log(e);
e=20;
console.log(e);
```
Output:
undefined
10
20
Explanation: -

1. **First console.log(e);**: At this point, e is declared but not yet assigned a value due to hoisting, so it outputs undefined.
2. **Second console.log(e);**: Now, e is assigned the value 10, so this outputs 10.
3. **After e = 20;**: The value of e is updated to 20.
4. **Third console.log(e);**: This outputs 20 since e was just assigned that value.

6) What will be the output of this code?

```
console.log(f);
var f=100;
var f;
console.log(f);
```
Output:
undefined
100
Explanation: -

1. **First console.log(f);**: Due to hoisting, the declaration var f; is hoisted to the top, so at this point, f is declared but not initialised, resulting in undefined.
2. **var f = 100;**: Here, f is assigned the value 100.
3. **Second console.log(f);**: Now that f has been assigned a value, this outputs 100.

The second declaration of f does not affect the value because var allows redeclaration without any issues.

7) What will be the output of this code?

```
console.log(g);
var g=g+1;
console.log(g);
```
Output:
undefined
NaN
Explanation: -

1. **First console.log(g);**: At this point, g is declared but not initialised, so it is undefined. The output will be undefined.
2. **var g = g + 1;**: Here, g is being referenced before it has been assigned a value. Since g is undefined, the expression g + 1 evaluates to NaN (Not-a-Number). This assignment means g is now NaN.
3. **Second console.log(g);**: Now, since g has been assigned NaN, this 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: -
Initially, the variable is declared but not assigned a value. As a result, when `console.log(h);` is executed, it outputs the default value of `undefined` .After that, when the value 50 is assign to `h`, a subsequent
call to `console.log(h);` will display the value 50.
9) What will be the output of this code?

```
console.log(i);
i=10;
var i=5;
console.log(i);
```
Output:
undefined
5
Explanation: -

1. **First console.log(i);**: The declaration var i; is hoisted to the top, so i is declared but not initialised yet, resulting in undefined.
2. **i = 10;**: This line assigns the value 10 to i, but since the assignment happens after the first console.log, it doesn't affect the first output.
3. **var i = 5;**: This line redeclares i and assigns it the value 5. Since i was already declared, this doesn't change the output of the first log.
4. **Second console.log(i);**: Now i is 5, so this outputs 5.