Variables in JavaScript

Variables:

Variables are used to store data in JavaScript. Variables are used to store reusable values. The values of the variables are allocated using the assignment operator (“=”). JavaScript Assignment Operator JavaScript assignment operator is equal (=) which assigns the value of the right-hand operand to its left-hand operand. JavaScript Identifiers.

Eg:

var a=49;

var b=10;

var h=5;

1. What will be the output of this code?

console.log(x);

var x=5;

Output:

Undefined

In JavaScript, the var declaration is hoisted, meaning it is moved to the top of its scope.

1. The first console.log(x); outputs undefined because x is declared but not yet assigned a value.
2. After the declaration, var x = 5; assigns 5 to x, but this occurs after the first log.

2. What will be output of this code?

console.log(a);

var a;

Output:

Undefined

In JavaScript, variable declarations using var are hoisted to the top of their scope.

1. The var a; declaration is hoisted, meaning it's recognized before any code execution begins.
2. However, since a is not assigned a value, it defaults too undefined.

3. What will be the output of this code?

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

Output:

Undefined

In JavaScript, the variable declaration with var is hoisted, but assignments are not.

1. The line console.log(b); attempts to log b before it has been declared or assigned a value.
2. At this point, b is undefined due to hoisting, so the output is undefined.
3. The assignment b = 10; occurs after the console.log, but it doesn't affect the output.

4. What will happen here?

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

Output:

Undefined Error

The output is undefined because the variable x is declared with var but not initialized before the console.log(x) statement. When JavaScript encounters the console.log, it finds that x has been declared (hoisted) but not yet assigned a value, resulting in undefined. Thus, undefined is printed to the console.

6. 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

The first console.log(e) outputs undefined because e is hoisted but not yet assigned a value. The subsequent logs output 10 and 20 as e is assigned the values in those lines.

7. What will be the output of this code?

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

Output:

Undefined

100

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.

8. What will be the output of this code?

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

Output:

Undefined

NaN

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`.

9. What will be the output of this code?

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

Output:

Undefined

50

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.

10. What will be the output of this code?

```
console.log(i);

i=10;

var i=5;

console.log(i);
```

Output:

Undefined

5

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.