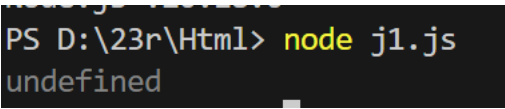


## Task

1. What will be the output of this code ?

```
console.log(x)

var x=5;
```

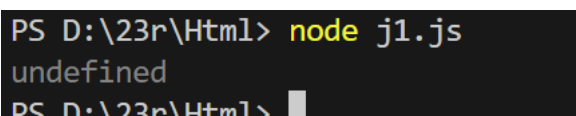
Ans:  PS D:\23r\Html> node j1.js  
undefined

JavaScript's interpreter processes the code in two phases: the creation phase and the execution phase. During the creation phase, all variable declarations (var) are moved to the top of their scope. However, only the declaration is hoisted, not the initialization.

2. What will be the output of this code ?

```
console.log(a);

var a;
```

Ans:  PS D:\23r\Html> node j1.js  
undefined

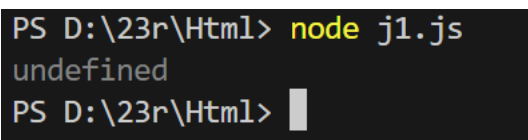
When the JavaScript engine processes the code, it moves the variable declaration to the top of its scope due to hoisting.

3. What will be the output of this code ?

```
console.log(b);

b=10;

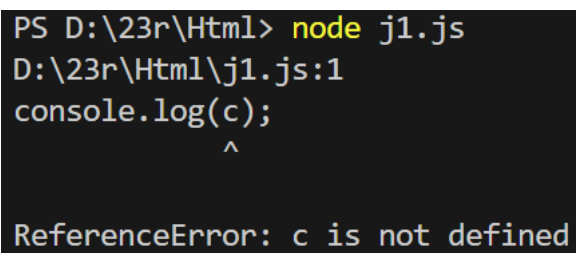
var b;
```

Ans:  PS D:\23r\Html> node j1.js  
undefined  
PS D:\23r\Html>

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

4. What will happen here?

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

Ans:  PS D:\23r\Html> node j1.js  
D:\23r\Html\j1.js:1  
console.log(c);  
 ^  
ReferenceError: c is not defined

the variable c has not been declared at all before you try to log it. Since there is no hoisting of any declaration for c, attempting to access it will result in a Reference Error.

5. What will be the output of this code?

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

```
PS D:\23r\Html> node j1.js  
undefined  
10  
20
```

Ans:

- The first console.log(e) will print undefined because the variable e is declared but not yet assigned any value.
- The second console.log(e) will print 10 because the variable e has been assigned the value 10.
- The third console.log(e) will print 20 after the value of e is updated to 20.

6. What will be the output of this code?

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

```
PS D:\23r\Html> node j1.js  
undefined  
100
```

Ans:

- The first console.log(f) will print undefined because the variable f is declared but not assigned a value due to hoisting.
- The second console.log(f) will print 100 since the value of f is assigned as 100 after the first log.
- The second var f; declaration has no effect because the variable is already declared.

7. What will be the output of this code?

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

```
PS D:\23r\Html> node j1.js  
undefined  
NaN
```

Ans:

- The first console.log(g) prints undefined because g is declared but not yet assigned any value.
- The statement g = g + 1 tries to add 1 to the value of g, but since g is undefined, this results in NaN (Not a Number).
- The second console.log(g) prints NaN.

8. What will be the output of this code?

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

```
PS D:\23r\Html> node j1.js  
undefined  
50
```

Ans:

- **Declaration of h:** The variable h is explicitly declared with var, so it exists in the scope.
- **First console.log(h):** Since the variable h is declared but not assigned any value, it prints undefined.
- **Assignment of h:** The variable h is then assigned the value 50.
- **Second console.log(h):** Now that h has a value, the second console.log(h) prints 50.

9. What will be the output of this code?

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

```
PS D:\23r\Html> node j1.js  
undefined  
40
```

Ans:

- **Variable Hoisting:** The declaration var i is hoisted to the top of its scope, but its assignment is not.
- **First console.log(i):** Since the variable i is declared but not yet assigned a value, the first console.log(i) prints undefined.
- **Assignment to i:** The value 10 is assigned to i right after the first log statement.
- **Reassignment to i:** The value of i is then reassigned to 40 immediately afterward.
- **Second console.log(i):** The second console.log(i) prints 40 because that is the current value of i.