Java Script Day 04

14 January 2024 01:01

Working on real life way of storing information for a website

Storing objects on an array

↓

later decompose the array & objects to get the req. data

↓

```
Const arrayinfo = [ { } , { } , { } , { } ]
```

↓

decomposing the objects inside array

↓

Here all Keys should be identical

```
let [ { Key : Var1, Key2 : Var2 } , , , { Key : Var3 , Key : Var4 } ] = arrayinfo
```

Console.log(Var1) Console.log(Var2) Console.log(Var3) Console.log(Var4)

↓ lengthy explanation

```
Const info = [ { name : "Sumit", class : 7 },
               { name : "Aman", class : 7 },
               { name : "Nitin", class : 9 } ]
```

Suppose if I want to know the information about Nitin's class

```
Const first_name = info [0]
let { name : nameVar } = first-name
Console.log(nameVar)
```

```
Const nitin_info = info [2]
let { class : classVar } = nitin_info
Console.log(classVar)
```

preferred way → for getting nitin's class

```
let { , , , { class : nitin_class } } = info
Console.log(nitin_class)
```

Functions in JavaScript

- named function
- function expression
- arrow functions

```
function name() {
    // Body of function
}
```

```
const function_name = function() {
    // Body of the function
}
```

```
const function_name = () => {
    // Body of function
}
```

example

```
const for_getindex = function(array, value) {
    for(let i in array) {
        if(array[i] === value) {
            return "value found at location ${i}"
        }
    }
    return "not found"
}
```

imp arrow function :

```
const arrowfun = (num1, num2) => num1 + num2
console.log(arrowfun(5,5))
```

Lexical Scope :

any function will first look for the values inside local scope & if not found will refer to the global scope

→ Complex JS programs utilize this technique for multiple dynamic reference or manipulation of a single variable.

Block Scope & Function Scope

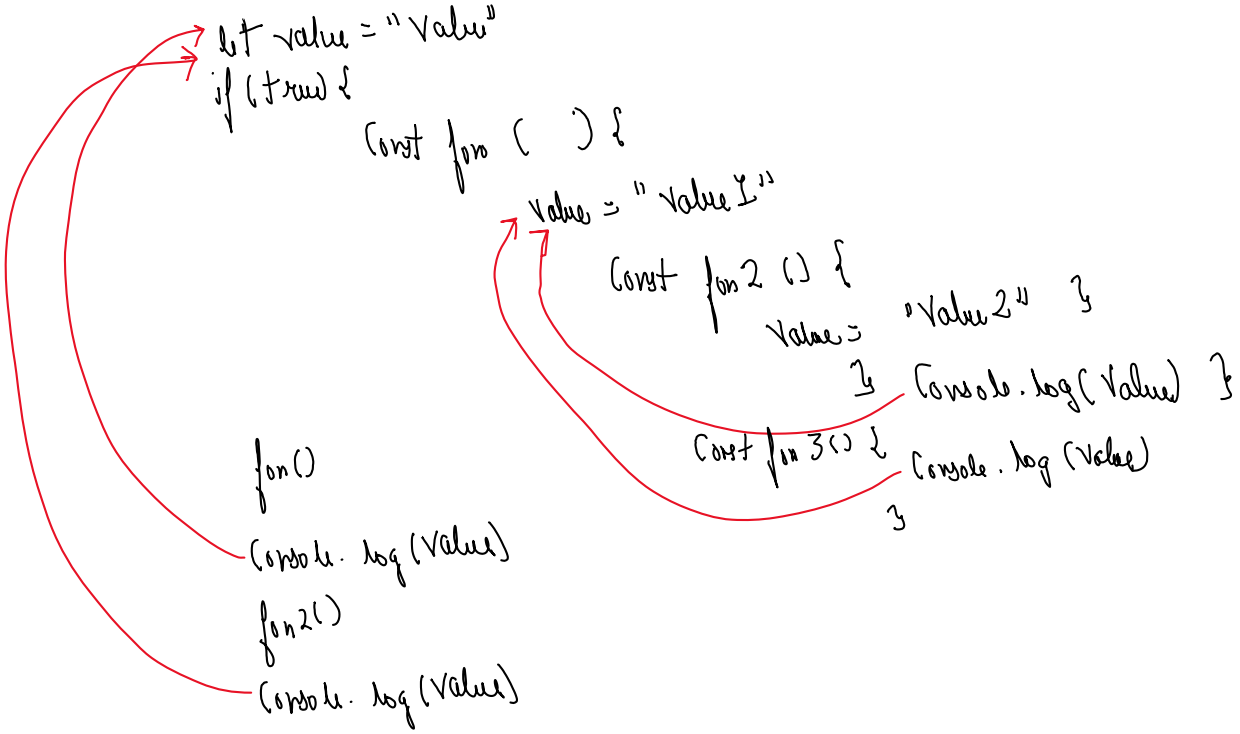
- Block Scope → let & const
  (You can only use them inside a single block)
- Function Scope → var
  (if declared inside any scope You can use variables - created with var anywhere in the program)

eg

```
let value = "value"
```

a single block

eg



default Parameters

```
const fun = (num1, num2 = 5) => num1 + num2
console.log(fun)
```

/

```
const fun = function(n1, n2)
{ if (typeof n2 === "undefined")
  { n2 = 5 }
  return n1 + n2
}
```

Rest parameters

↓

Convert rest of the parameters into an array

↓

eg

```
const funSum = (n1, n2, ... n3) => { sum = 0
  for (let i in n2) {
    sum = sum + i
  }
  return sum * (n1 + n2)
}
console.log( funSum( 2, 4, 6, 8, 10, 12, 14, 16, 18, 20) )
```

n1 ← stored in n1

n2 ← stored in n2

... n3 ← converted into array & stored in ... n3

destructuring Parameters :

↓

Using obj destructuring inside parameters of a function

↓

eg

```
object_dst = { name : "ravi",
               company : "Pluralsight" }
```

eg object_dest = { name : "ravi",
company : "PureIT" }

```
const objfundest = ( { name : valname , company : valcomp } ) => {
                    console.log ( valname )
                    console.log ( valcomp ). }

objfundest ( object_dest )
```

Callback function → a function which is dynamically passed inside a function but later we can update its value dynamically

↑

Higher Order function

↓

function returning function → a function which will dynamically returns the functions

eg

Callback

```
const fun = ( name ) => {
    return " name of student is
    ${name} "
}

const callback = ( callback ) => {
    // taking name from the database
    console.log ( callback ( names_from_sql ) )
}

callback ( fun )
```

function returning function

```
const fun = function ( ) {
    const infun = function () {
        console.log ("inside fun")}
    return infun ;
}

const a = fun()
a ();
```