Java Script Day 3

13 January 2024 02:05

Working With Primitive & Reference data types :

Primitive ↓

if this data var gets updated then the variable pointing to the previous data type will not get updated

example :

```
let a = 5
let b = a
a = a+5
console.log(a)   → o/p = 8
console.log(b)   → o/p = 5
```

Reference →

this data type will contain a reference location along with values so if this data var get updated all the data types which is pointing to the above data type will also get updated.

example :

```
let array = [1,2,3,4,5]
let array1 = array
array.push(9)
console.log(array)   → [1,2,3,4,5,9]
console.log(array1)  → [1,2,3,4,5,9]
```

Both will produce same o/p

How to copy the Reference Data type

Copying an array :

→ array.slice(0) + []

↓ [].concat(array)

↙ [... array]

↓ most used spread operator (must be inside list)

iterating array variables present inside loops :

```
const array = [1,2,3,4,5]
let array2 = []
```

for loop ← returns iterate

```
for(let i=0; i<array.length-1; i++)
{
array2.push(array[i].toUpperCase())
}
```

for of loop → return element as output

for in loop → return index.

```
for(let index in array)
{
console.log(array[index]
```

[illegible]

}

↓

```
for ( let arr of array )
{   console . log ( arr )
}
```

console.log (array [index]
)

Decomposition of Array :

Saving elements of array into seperate variables : ⇒ multiple assigning of data inside an array to a new variables

```
const array = [ 1,2,3,4,5,6,7 ]

let [ 1st element , 2nd Element , , 4th element , " " , "6th element" ] = array
      ⇓
      OP →   1st element = 1
             2nd element = 2
             4th element = 4
             6th element = 6
```

Console.log ( ) ↗

And

```
let [ element 1 , element 2 , ...Slied array ] = array
```

↗ Here O/P

element 1 = 1
element 2 = 2
Slied array = [3,4,5,6,7]

Objects :

These are same as dictionaries in python.

```
const object1 = { key1 : value1 ,
                  key 2 : value2
                }
```

→ For accessing values : ↗ (same as dictionary)

✓ object1 [key1] ✓✓ prefered since is the only method that support spaces

object1 . key)

iterating objects using loops :

(1) →

```
for ( let i in object1 )
{
  console . log ( ` The value in key ${ i } is ${ object1 [ i ] } ` )
}
```

(1) →

```
const keysInObj = Object . keys ( object_Name )
console . log ( key In obj )
```

This will return the array of keys in output

another way of initializing dynamic variables to the obj ↙

Variables to the obj ↓

```
const Key = "K"
const Key1 = "v"
objN = {
        [Key]  : "Keys"
        [Key1] : "Keys1"
        }
```

Square Brackets.

the array of Keys in output

Spread Method for Objects Manipulation

```
obj1 = { N: "name",
         P: "position" }
obj2 = { C: "Class",
         R: "Regions",
       }
combine_obj = { ...obj1 , ...obj2 }
```

Object Decomposition :

```
const object = {
        name : 'a'
        gender : 'm',
        pos : 'p' }  →  let { name , pos } = object
                        console.log (name)    console.log (name)
                          ↓ op                  ↓ op
                          a                     p
```

[if You want to save the Key's value to another var_name You can do this] ↘

```
let { name : var1 , pos : varP } = object
console.log (var1)        console.log (varP)
op ↓                        ↓ op
  a                          p
```

decomposition to both Var & object

↓

```
const { Key1 : Val1 , Key2 : Val2 , ... remaingKeys } = object
```

Outputs ↓

```
console.log ( Val1)  ↗ Value
console.log (Val2) ↘ Value
console.log ( remainingKeys) ↘ object
```