Tuesday, 29 August 2023 9:57 PM

0

```
4  let count = 0;
5  (function printCount() {
6    if (count === 0) {
7       let count = 1;
8       console.log(count);
9    }
10    console.log(count);
11  })();
12
```

→ So off will be: 10

→ Since let is block scoped so line & will be console I as as another let variable will be areated for the block which will shadow the global variable.

- After the block is completed, the inside let variable gets destroyed, so, since & forms a closere with global saape iso o will be printed.

Q' Write a pr that would allow you to do that:

```
var addSix = createBase(6);
addSix(10); // returns 16
addSix(21); // returns 27
```

01/8.

```
function createBase(num) {
  return function (innerNum) {
    console.log(innerNum + num);
  };
}
```

forms a closure with createbase() littles, has access to its parameters.

-> Time optimization!

```
36 <u>script.js:10</u>
6: 57.974853515625 ms <u>script.js:15</u>
144 <u>script.js:10</u>
12: 64.512939453125 <u>script.js:18</u>
ms

> |
```

-> This code takes much time to excute.

To optimize this we can use dosure, so that the loop & let Variable remains the same for every time we call dosure ().

```
4 function find() {
5    let a = [];
6    for (let i = 0; i < 1000000; i++) {
7       a[i] = i * i;
8    }
9
10    return function (index) {
11       console.log(a[index]);
12    };
13 }</pre>
```

36	script.js:11
გ: 0.25 ms	<u>script.js:18</u>
2500	<u>script.js:11</u>
50: 0.02587890625 ms	script.js:21

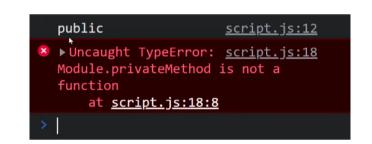
-250, we've optimized this

```
const closure = find();
console.time("6");
closure(6);
console.timeEnd("6");
console.time("50");
closure(50);
console.timeEnd("50");
```

- Basically we store the closure of the retrurned for from And() in closureco, so now we do not recreate that loop whenever dosureco) is called.

Note: We have a convention to create private un'ables with '_ ' (undersone).

- Module puttern:



TIFE & has a private method a public method.

-> A public method is accessible situate the

module pattern but not a private method-

I the private methods can only be accessed by the public method.

> Make a for that can only be called once; > To do this we use the soncept of closures.



-> 50, here we make a f object that returns on anonymous f which makes a dosure with once().

> Now we store the returned for dosure in called the Kinethons), so, the instance of called variable is not created again. If we had done once()()', then it would not have wrorked because for each time once()()' would have run, alled variable would have Thirthdised.

-> We have oncest for in Lodash library as well



```
function once(func, context) {
    let ran;

    return function () {
        if (func) {
            ran = func.apply(context || this, arguments);
            func = null;
        }

        return ran;
};

const hello = once(() => console.log("hello"));

hello();
hello();
hello();
hello();
hello();
hello();
hello();
```

(once () poly fill)

```
function once(func, context) {
    let ran;
    return function () {
        if (func) {
            ran = func.apply(context || this, arguments);
            func = null;
            return ran;
        };
}

const hello = once((a, b) => console.log("hello", a, b));

hello(1, 2);
hello(1, 2);
hello(1, 2);
hello(1, 2);
hello(1, 2);
```

(when for takes in some arguments)

> So, we create a f" 'oncecs' which take 2 arguments, a callback f" La context (for telling this keyword where it should point to). Once of f" returns a f" which will make the callback f" only run once.

+ Returned for forms obsure with 'once W'. We dedone ran outside the returned for because we do not want van to be initialized everything the returned for is called.

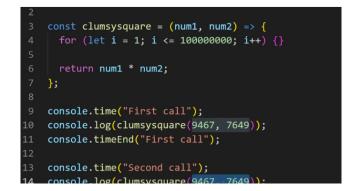
I Now for the 1st time, func will always be true, since we are passing the cultback of to sheec). Thus, if and fin will run, and we will use apply method to run the f'as we need to pass water to Larguments as a list.

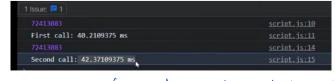
-> Now, it context is not passed through oncess then we will pass this ' keyword which will point to the local memory of the returned f".

The will also pass the arguments, arguments is an array like object which has the arguments possed to at a is outo nationly ownilable in case of normal tis but is not available in case of arrow fis. I finally we set func shull so that if and it is not run again.

-> We store the closure of the returned for in hello () & pass the callback of to & noe() . Now however many times we call hello () it will only run once.

> Implementing memoize/ceeling pho





ofp when the fhis called with some arguments twice.

If a f takes a good amount of time to vun then we must save its value somewhere, so that if the same arguments are passed, it uses the memoirzed valued bes not take the same amount of time as before.

```
function myMemoize(fn, context) {

const res = {};

return function (...args) {

var argsCache = JSON.stringify(args);

if (!res[argsCache]) {

res[argsCache] = fn.call(context || this, ...args);

return res[argsCache];

return res[argsCache];

};

}
```

(memoize caching Fn)



(olp of memoization which is far better than better)

```
res = {
  "5,6": 30,
};
```

Chow the res
object will look
with de hamic key; value
yours

```
const clumsyProduct = (num1, num2) => {
  for (let i = 1; i <= 1000000000; i++) {}

  return num1 * num2;
};

const memoizedClumzyProduct = myMemoize(clumsyProduct);

console.time("First call");

console.log(memoizedClumzyProduct(9467, 7649));

console.timeEnd("First call");

console.time("Second call");

console.time("Second call");

console.log(memoizedClumzyProduct(9467, 7649));

console.timeEnd("Second call");</pre>
```

(calling the original for by passing it to the memorizing for)

→ Our my Memoize() & takes in a callback & forms closure with

The returns a memoized & which takes in arguments & forms closure with

its parent & we bedove here subside the returned of so that it is

not created again & again when inner or is returned. Otherwise since

const is block scaped, a new value of her will be created when the inner of is

resturned which would be an empty object.

Now, we convert the ... args received to a string using Ison stringit.

Listore it in ourgs Cashe.

I Now 'args Cache' will be the dynamic key in res' object. So if restargs Cache] is there in res' object then we'll return it otherwise we'll call the collback of using call() method & pas the context '(if received otherwise pas'this keyword which points to the inner of) & pass the arguments using - args & store the returned value in wes' object & Finally return restargs Cache].