Date: / /

Closures in JavaScript

```
function sayWord (word) {
    return () => console.log(word);
}
const sayHello = SayWord ("hello");

SayHello(); // "hello"
```

There's 2 interesting point to notice :-

→ The returned function from sayWord can access the word parameter

→ The returned function maintain the value of word when SayHello is called outside scope of word.

The first point can be explained by Lexical Scope. :

Lexical Scope - The returned function can access word before it exists in its outer scope

Date: / /

The second point becoz of Closures

A closure is a function combined with references to variables define outside of it.

Closure maintain the variable references, which allow function to access variables outside of their scope.

They "enclose" the function and variable in its environment

Example of Closures In JavaScript

Callbacks - It is common for a callback to reference a variable declared outside of itself

eg → function getCars By Make (make) {
return cars.filter(x => x.make === make);
}

make is available in callback because of lexical Scoping and make is persisted when filter called becoz of closure

Date: / /

Storing state → We can use closures from functions that store states

Let's say a fn which returns an object that can store and change name.

```
function makePerson(name) {
    let _name = name;

    return {
        setName: (newName) => (_name = newName),
        getName: () => _Name,
    };
}

const me = makePerson("deepa");
console.log(me.getName()); // "deepa"

me.setName("Deepa Chaurasia");
console.log(me.getName());
                // "Deepa Chaurasia");
```

It shows how closure do not freeze values of variables from function's outer scope during creation. Instead they maintain the references throughout closure's lifetime.

Date: / /

Private methods

So In oops concept, In a class we have private state and expase getter and setter methods public.

We can extend this oops.

```
function makePerson(name) {
    let _name = name;

    function PrivateSetName(newName){
        _name = newName;
    }

    return {
        setName: (newName) => privateSetName
                              (newName),
        getName: () => _name,
    };
}
```

Date: / /

PrivateSetName is not directly accessible to consumers and it can access the private state variable - name through closure.

Closures make it possible for:

functions to maintain connections with outer variables, even outside scope of the variables

(like LinkedIn maybe :))

There are many uses of closures from creating class like structures that store state and implement private methods to passing callback to event handlers.