



# Class - 3

\* Call, bind & apply

\* prototypes & prototypical inheritance

\* Call method

let obj = {  
 abc() {  
 // do something  
 }  
}

\* abc is implicitly binded  
obj

function abc() {

}

let obj = {

}

\* method is a function, dataStructure.method();

---

\* call is a method of function

\* abc.call(obj);

① It will attach abc inside obj,

② It will execute abc!  $\Rightarrow$  obj.abc();

③ delete abc from obj.

---

function abc (param1, param2) {

==

}

abc (arg1, arg2);

function abc ( param1, param2 ) {

}

---

\* abc ( 1, 2, 3, 4, 5 );

\* function abc ( param1 , param2 ) {  
    // do something  
}

Let obj = { }

\* abc ( arg1 , arg2 );  
\* abc.call ( obj , arg1 , arg2 );

\* abc ();  
\* abc.call ( obj );

function test (param1, param2, param3) {  
 let obj,  
 ~~~~~  
 ~~~~~  
 ~~~~~  
}

---

\* arg1, arg2, arg3

---

\* test (arg1, arg2, arg3) → test.call (obj, arg1, arg2, arg3)

---

```
function test1 ( param1, param2, param3 ) {  
  // doing something.  
}  
let obj =  
{ ... };  
}
```

---

```
let args = [ 10, 20, 100 ];
```

---

```
* test1.call (obj, args[0], args[1], args[2]);
```

---

```
* test1.call (obj, ...args);
```

```
* test1.apply (obj, args);
```



\* functionName.call (objectContext, arg1, arg2, ----);

---

Call can accept multiple args

first is obj, rest are arguments of function

---

\* functionName.apply (objectContext, array of arguments);

apply can accept only 2 arguments

1) obj

2) array of arguments for function

\* call & apply have 1 issue??

\* function  $f!$ , obj!

momentarily

\*  $f!.call(obj!)$

---

\* I will make new fn out  $f!$ ,  
that is having def of  $f!$  & context  
of  $obj!$

---

\* `const f2 = f1.bind(obj);`



It is a function that is having def<sup>n</sup> of `f1` & value of this key word inside `f2` will be `obj`,

---

\* function f1 (param1, param2, param3) {

\* let obj = { };

}

① f1 (arg1, arg2, arg3);

\* const f2 = f1.bind(obj);

① \* f2 (arg1, arg2, arg3);

function f1 (param1, param2, param3) {  
 let obj = { };  
 \_\_\_\_\_  
}

const f2 = f1.bind (obj, arg1);

\* f2 (param2, param3) {  
 param1 = arg1;  
}

}

function f1 (p1, p2, p3) {

let obj = { 3; }

\*

{

\* f2 = f1. bind (obj, arg1, arg2);

\* f2 (100, 200);