Advanced JavaScript

•••

Lecture 2

This - What is This keyword ?

- This is references the Object is executing the current function.
- If we are inside method (Function inside an object) , the This value will refers to the current Object.
- If we are inside regular function (Not an Object) , the This value will refers to the global Object (Window Object).
- Example: "This" inside Method / Function

The use of This in the global scope

- ❑ In the global scope, when the code is executing in the browser, all global variables and functions are defined on the window object. Therefore, when we use this in a global function, it refers to (and has the value of) the global window object).
- ❑ **Remember:** window is the object that all global variables and functions are defined on.

this - when used in a method passed as a callback

- when we pass a method (that uses this) as a parameter to be used as a callback function, the value of this will not be as expected.
- the this keyword no longer refers to the original object where "this" was originally defined, but it now refers to the object that invokes the method where this was defined.
- Example: this in a method passed as a callback

this - when method is assigned to a variable

- The this value is bound to another object, if we assign a method that uses this to a variable.
- In this case : this will refer to the window object
- Example: this - when method is assigned to a variable

JavaScript Bind() Method

- We use the Bind () method primarily to call a function with the this value set explicitly.
- In other words, bind () allows us to easily set which specific object will be bound to this when a function or method is invoked.
- Example: this in a method passed as a callback

JavaScript call() , apply() Method

- ❑ they allow us to set the this value in function invocation.
- ❑ the apply() function in particular allows us to execute a function with an array of parameters.
- ❑ the call() function in particular allows us to execute a function with an list of parameters separated by comma.
- ❑ Example: call() method
- ❑ Example: apply() method

Nested functions

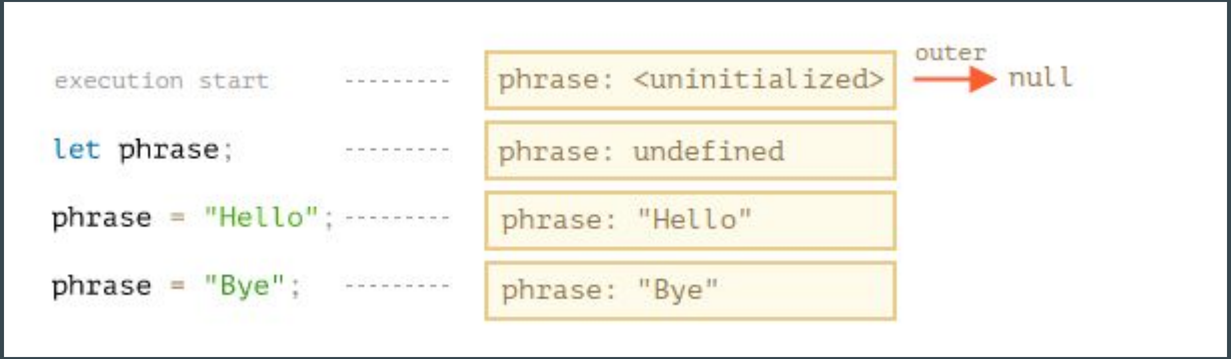
- A function is called “nested” when it is created inside another function.
- The inner function It can access the outer variables and so can return them.
- Example: nested functions
- Example: closure counter example

Lexical Environment

- In JavaScript, every running function, code block {…}, and the script as a whole have an internal (hidden) associated object known as the Lexical Environment.
- The Lexical Environment object consists of two parts:
    - Environment Record – an object that stores all local variables as its properties (and some other information like the value of this).
    - A reference to the outer lexical environment, the one associated with the outer code.

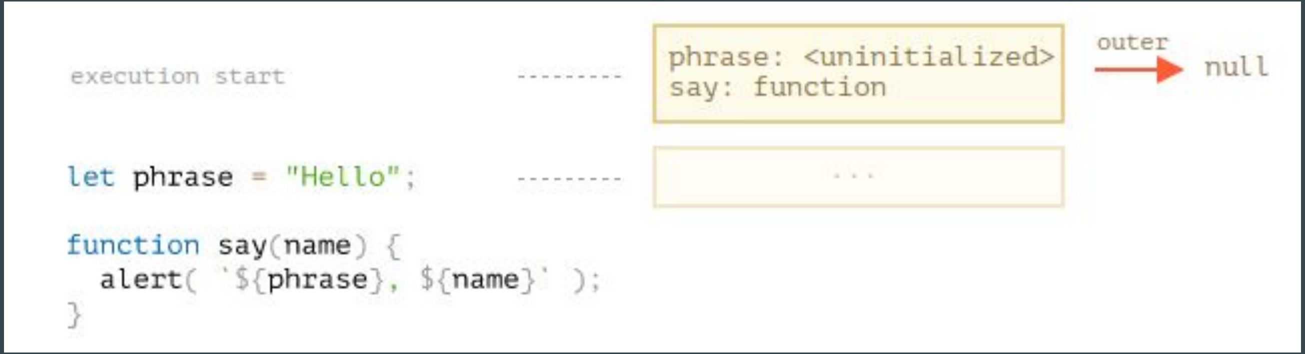
Lexical Environment - Variables

- A "variable" is just a property of the special internal object, Environment Record. "To get or change a variable" means "to get or change a property of that object".


```
execution start     ---------   phrase: <uninitialized>
let phrase;         ---------   phrase: undefined
phrase = "Hello";   ---------   phrase: "Hello"
phrase = "Bye";     ---------   phrase: "Bye"
```

Lexical Environment - Functions

- When a Lexical Environment is created, a Function Declaration immediately becomes a ready-to-use function
- That’s why we can use a function, declared as Function Declaration, even before the declaration itself.

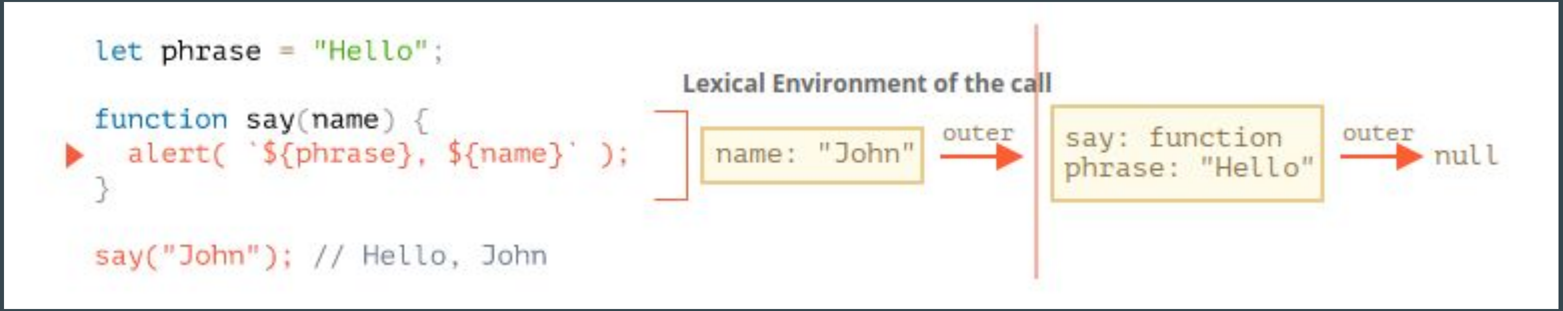


```
let phrase = "Hello";

function say(name) {
  alert( `${phrase}, ${name}` );
}
```

Inner and outer Lexical Environment

- When a function runs, at the beginning of the call, a new Lexical Environment is created automatically to store local variables and parameters of the call.



```
let phrase = "Hello";

function say(name) {
  alert( `${phrase}, ${name}` );
}

say("John"); // Hello, John
```

Inner and outer Lexical Environment

- During the function call we have two Lexical Environments: the inner one (for the function call) and the outer one (global).
- The inner Lexical Environment has a reference to the outer one.
- When the code wants to access a variable – the inner Lexical Environment is searched first, then the outer one, then the more outer one and so on until the global one.

Closure

- ❏ A closure is a function that remembers its outer variables and can access them.
- ❏ in JavaScript, all functions are naturally closures (there is only one exception, to be covered in The "new Function" syntax).