

Actividad Sesión 2

Alumno: Esteban Antonio Castro Rojas

Repositorio: <https://github.com/esancaro/Javascript-Course>

Fecha: 2022-07-05

Ejercicio 1

Escribe un programa que tome como entrada un objeto y devuelva una lista con sus propiedades. Solo puede tener como entrada un objeto y el tipo de vuelta tiene que ser un array.

Código:

```
// Ejercicio 1
let person = {
  name: "Esteban",
  age: 47,
  profession: "Developer Wannabe",
}

console.log(Object.keys(person));
```

Salida:



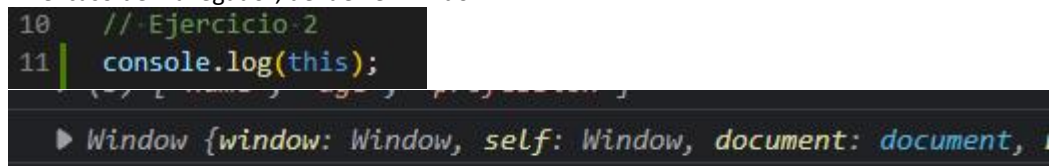
```
(3) ['name', 'age', 'profession']
```

Ejercicio 2

Enumera los distintos valores que puede tener *"this"* y pon un ejemplo de cada uno.

1. Cuando se ejecuta en programa principal, devuelve el objeto global.

a) En el caso del navegador, devuelve "window".



```
10 // Ejercicio 2
11 console.log(this);

Window {window: Window, self: Window, document: document, ...}
```

- b) En el caso de nodejs, devuelve el objeto global.

```
node
~ node
Welcome to Node.js v16.15.1.
Type ".help" for more information.
> this
<ref *1> Object [global] {
  global: [Circular *1],
  clearInterval: [Function: clearInterval],
  ...}
```

2. Ejecutada en una función:

- a) Devuelve el objeto global.

```
// Ejercicio 2
1 console.log(this);
2
3 // 2.2a
4 function fFoo() {
5   return this;
6 }
7
8 console.log(fFoo());
```

Window {window: Window, self: Window, ...}

- b) En modo estricto, devuelve "undefined".

```
// 2.2b
20 function fFooStrict() {
21   "use strict";
22   return this;
23 }
24
25 console.log(fFooStrict());
```

undefined

3. En un método:

- a) "This" apunta al objeto propietario.

```
// 2.3 this en un método
7 let fooObj = {
8   foo: "lorem ipsum",
9   bar: function() {
10     console.log(`2.3. This foo: ${this.foo}`);
11   }
12 }
13
14 fooObj.bar();
```

2.3. This foo: lorem ipsum

- b) En un arrow function, this no se enlaza con el objeto que la llama, siempre representa el objeto que definió la función. [JavaScript Arrow Function \(w3schools.com\)](https://www.w3schools.com/js/js_arrow_functions.asp).

```
// 2.3 this en un método: function y arrow
let fooObj = {
  foo: "lorem ipsum",
  bar: function(){
    console.log(`2.3. This foo: ${this.foo}`);
  },
  baz: () => {
    console.log(`2.3b. This is foo: ${this.foo}`)
  },
  qux: () => {
    let foo = "dolor sit amet"; // esto lo entendí mal
    console.log(`2.3b-1. This is foo: ${this.foo}`)
    console.log(`2.3b-1b. This is foo: ${this}. This is pointing to window.`)
  },
  ctrlF: function(){
    console.log(`Function this: ${this}`);
  },
  ctrlA: () => {
    console.log(`Arrow this: ${this}`);
  }
}
fooObj.bar();
fooObj.baz();
fooObj.qux();

document.getElementById("ej2-bf").addEventListener("click", fooObj.ctrlF);
document.getElementById("ej2-ba").addEventListener("click", fooObj.ctrlA);
```

Ejercicio 2

👉 Enumera los distintos valores que puede tener "this".

👉 Prueba para function y arrow:

2.3. this foo: lorem ipsum

2.3b. This is foo: undefined

2.3b-1. This is foo: undefined

2.3b-1b. This is foo: [object Window]. This is pointing to window.

4. Bind, call y apply le vinculan "this" al objeto que uno pasa por el parámetro. Bind realiza la vinculación, call vincula y llama a la función pasando una lista de parámetros, apply hace lo mismo pasándole un arreglo.

```

54 function fooBind(){
55   console.log(`Bind Test: ${this.foo}. This is ${this}`);
56 }
57
58 funcBind = fooBind.bind(fooObj);
59 funcBind();
60
61 function fooBindWParams(param1, param2){
62   console.log(`Bind Test: ${this.foo}. This is ${this}. param1: ${param1}, param2: ${param2}`);
63 }
64
65 fooBindWParams.call(fooObj, 'param one', 'param two');
66 fooBindWParams.apply(fooObj, ['Array 1', 'Array 2'])
67
68 Bind Test: lorem ipsum. This is [object Object]
69 Bind Test: lorem ipsum. This is [object Object]. param1: param one, param2: param two
70 Bind Test: lorem ipsum. This is [object Object]. param1: Array 1, param2: Array 2

```

Ejercicio 3

Las cadenas son inmutables, la mejor opción es realizar un split, reverse, join.

Código de la solución (como aparece en la consola de /activity/2_session/index.html)

```

// una clase a la que llamaremos "InvertirCadena"
class InvertirCadena {
  // Un atributo llamado cadenaInvertir que sea una cadena vacía.
  cadenaInvertir = '';
  // Una función en formato arrow function, que tome el atributo
  // cadenaInvertir
  // e imprima en pantalla el resultado invertido
  do = () => {
    if (this.cadenaInvertir == '') throw "Cadena no definida.";
    return this.cadenaInvertir.split('').reverse().join('');
  }
}

// Ahora instancia la clase en un objeto que llamaremos invertirCadena
let invertirCadena = new InvertirCadena();

```

```

try{ // ¿Cómo podemos hacer para que nuestro código no rompa al
ejecutarse?
  invertirCadena.do();
} catch (e) {
  console.log(e);
}

```

```

// cambia el valor a cadenaInvertir y vuelve a llamar la función
invertirCadena.cadenaInvertir = "Hola Mundo!";
console.log(invertirCadena.do());

```

```

// intenta acceder al siguiente método invertirCadena.nuevoMetodo()

```

```
invertirCadena.nuevoMetodo?.(); // optional chaining
```

Salida:

```
Cadena no definida.
```

```
!odnuM aloH
```

Ejercicio 4

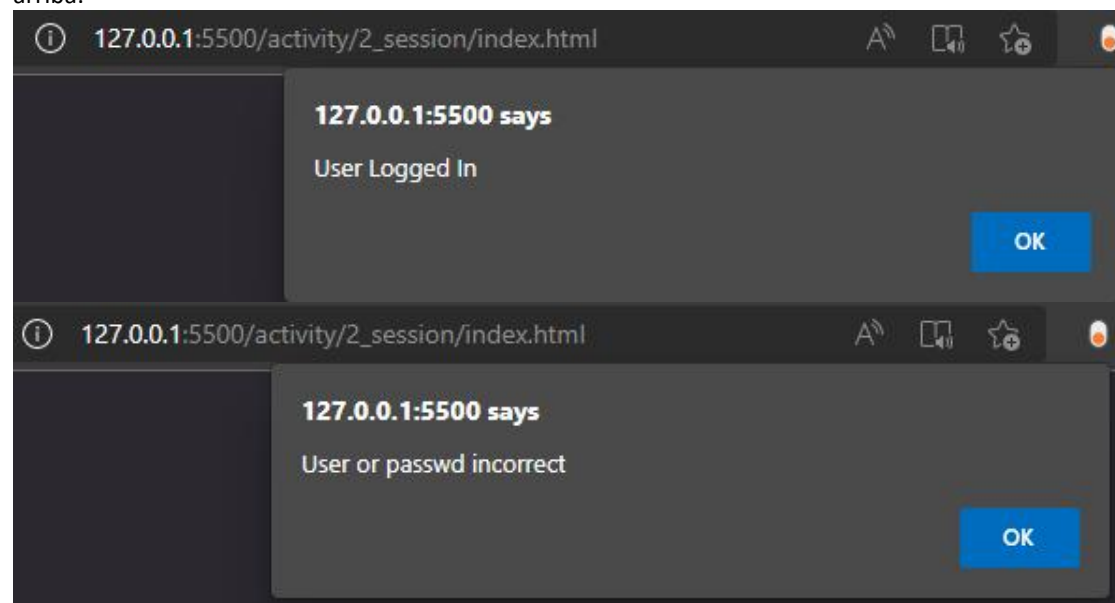
El siguiente código despliega una alerta al crear la clase login.

```
class Login {
  constructor(username, password){
    this.username = username;
    this.password = password;
    this.login();
  }

  login = function() {
    console.log(this.username, this.password);
    if(this.username == "admin" && this.password == "passwd"){
      window.alert("User Logged In");
    } else {
      window.alert("User or passwd incorrect")
    }
  }
}
```

```
let login = new Login("admin", "passwd");
let badLogin = new Login("esteban", "n/a");
```

Salida, en el navegador. /activity/2_session/index.html. Para ambos casos, como aparece el código de arriba.



Ejercicio 5

Se han comentado los alerts del ejercicio previo. Se agregaron event listeners a los botones.

Código:

```
// let login = new Login("admin", "passwd");  
// let badLogin = new Login("esteban", "n/a");  
  
document.getElementById("loginSuccess").addEventListener("click", () =>  
  new Login("admin", "passwd"));  
document.getElementById("loginFailure").addEventListener("click", ()  
=> new Login("esteban", "n/a"));
```

Ejercicio 6

Se agregaron event listeners a los botones "async":

Código:

```
document.getElementById("loginSuccessAsync").addEventListener("click",  
(() => loginWithUsername("admin", "passwd")  
  .then((resolve) => {alert(resolve)})  
  .catch((rejected) => {alert(rejected)}))  
);  
document.getElementById("loginFailureAsync").addEventListener("click",  
(() => loginWithUsername("esteban", "n/a")  
  .then((resolve) => {alert(resolve)})  
  .catch((rejected) => {alert(rejected)}))  
);
```

Salida, en el navegador. /activity/2_session/index.html.

