Funciones

En Javascript una función es un procedimiento, un conjunto de sentencias que realizan una tarea o calculan un valor. Para usar una función se debe definir en algún lugar del ámbito desde el cual se desea llamarla.

Declaración de función

Para declarar funciones vamos a usar la siguiente sintaxis:

```
function nombre([param[,param[, ...param]]]) {
    instrucciones
}
```

Ejemplo:

```
function displayName(name) {
    console.log(name);
}
```

return

La sentencia return **finaliza** la ejecución de la función y especifica un valor para ser devuelto a quien llama a la función.

```
function displayName(name) {
    return "Mi nombre es " + name;
}
```

Ejecución de funciones

Para ejecutar una función que anteriormente hayamos creado, basta con llamarla como: nombre_de_funcion(params);

El valor de la función puede ser asignado a una variable para luego utilizarlo.

Ejemplo:

```
function displayName(name) {
    return "Mi nombre es " + name;
}

const name = displayName("CourseIT");
console.log(name); //CourseIT
```

Ámbito de una función

Las variables definidas dentro de una función no pueden ser accedidas desde ningún lugar fuera de la función, ya que la variable está definida sólo en el ámbito de la función. Sin embargo, una función puede acceder a todas las variables y funciones definidas dentro del ámbito en el cual está definida.

```
function display(name) {
    const textName = "Mi nombre es " + name;

    function addName() {
        return textName + "IT";
    }

    return addName();
};

const name = display("Course");
console.log(name); //CourseIT
```

Expresiones de función

Las funciones pueden también ser creadas por una expresión de función, la cual puede ser anónima, por ejemplo, tomando el caso anterior de la función de displayName, nos quedaría así:

```
const displayName = function(name) {
    return "Mi nombre es " + name;
}

const name = displayName("CourseIT");
console.log(name); //CourseIT
```

Expresiones de función

Sin embargo, se puede proporcionar un nombre a una expresión de función y éste puede ser utilizado dentro de la función para referirse a si misma, vamos a usar de ejemplo una función que calcula el factorial de un entero positivo (El factorial de un entero positivo es el producto de todos los números enteros desde 1).

```
const factorial = function fac(number) {
    if (number < 2) {
        return 1;
    } else {
        return number * fac(number - 1);
    }
};

factorial(5);
```

Clousure o Clausura

Los closures o clausuras son funciones que manejan variables independientes. En otras palabras, la función definida en el closure "recuerda" el ámbito en el que se ha creado.

Métodos privados usando Clousures

Vamos a crear un clousure donde la variable value sea privada y solo pueda ser accedida mediante funciones publicas.

```
function counter() {
    let value = 1;

    return {
        incrementValue: function() {
            value = value + 1;
        },
        decrementValue: function() {
            value = value - 1;
        },
        value: function() {
            return value;
        }
    };
}

const newCounter = new counter();
newCounter.value(); //1
newCounter.incrementValue();
newCounter.value(); //2
newCounter.decrementValue();
newCounter.value(); //1
```

Clousures públicos

Usando la variable “this” podemos hacer publico las funciones o variables que agreguemos en nuestro clousure, usando el ejemplo anterior, nos quedaría de la siguiente manera

```
function counter() {
    this.value = 1;

    this.incrementValue = function() {
        this.value++; //Podemos usar ++ en reemplazo del
                      //this.value = this.value + 1
    };

    this.decrementValue = function() {
        this.value--; //Podemos usar -- en reemplazo del
                      //this.value = this.value - 1
    };
}

const newCounter = new counter();
newCounter.value; //1
newCounter.incrementValue();
newCounter.value(); //2
newCounter.decrementValue();
newCounter.value; //1
```

Clases

Las clases de Javascript llegaron con ES6 (ECMAScript 2015), son una mejora sintáctica sobre la herencia basada en prototipos de JavaScript. La sintaxis de las clases no introduce un nuevo modelo de herencia orientada a objetos en JavaScript. Las clases de JavaScript proveen una sintaxis mucho más clara y simple para crear objetos y lidiar con la herencia.

Definiendo clases

Para la declaración de una clase, es necesario el uso de la palabra reservada class y un nombre para la clase:

```
class Counter {
    constructor(startValue) {
        this.value = startValue;
    }
}
```

El método constructor es un método especial para crear e inicializar un objeto creado con una clase. Solo puede haber un método especial con el nombre "constructor" en una clase. Si esta contiene mas de una ocurrencia del método constructor, se arrojará un Error SyntaxError

Métodos

Dentro de los métodos a usar en las clases tenemos 2, prototipo y estáticos, los prototipo se llaman cuando la clase es instanciada (mediante el new)

```
class Counter {
    constructor(startValue) {
        this.value = startValue;
    }

    getValue() {
        return  `El valor es ${this.value}`;
    }

    incrementValue() {
        this.value++;
    }

    decrementValue() {
        this.value++;
    }
}

const newCounter = new Counter(0); //El 0 es el startValue
newCounter.getValue(); //El valor es 0
newCounter.incrementValue();
newCounter.getValue(); //El valor es 1
newCounter.incrementValue();
newCounter.getValue(); //El valor es 2
newCounter.decrementValue();
newCounter.getValue(); //El valor es 1
```