ACADEMY by NUMEN

FULL STACK

Comenzamos en unos minutos


ACADEMY
by NUMEN

Javascript: Clase 2
Tipos de datos compuestos y Bucles

ACADEMY by NUMEN

Toma de decisiones en Javascript

Primitivos: Se accede **directamente** al valor.

- string (cadena de textos)
- number
- boolean (true, false)
- null
- undefined
- NaN

Compuestos: Se accede a la **referencia** del valor.

- object = {}
- array = []
- function() {}
- Class {}
- etc

Tipos de datos compuestos: Arreglo

Los *arreglos* son objetos similares a una lista cuyo prototipo proporciona métodos para efectuar operaciones de recorrido y de mutación. Tanto la longitud como el tipo de los elementos de un *array* son variables.

Los métodos aplicables a los arreglos los veremos de manera más detallada en próximas diapositivas. Un arreglo se ve más o menos así:

```
var miPrimerArreglo = ["Hola", 1, true, ["a", "b", "c"]]

console.log(miPrimerArreglo)

console.log(miPrimerArreglo.length)
```

Acceso a elementos en una matriz

Para acceder al elemento, escribiremos el nombre o la variable de matriz, seguidos de corchetes que contienen la asignación numérica.

Los elementos reciben una posición numérica (índice) de acuerdo con su ubicación en la matriz, en orden. El orden numérico de una matriz SIEMPRE comienza en 0, por lo que el primer elemento está en el índice 0, el segundo en el índice 1, el tercero en el 2, y así sucesivamente

```
var estudiantes = ['Martin', 'Fernando', 'Sara', ['Samuel', 'Jorge',
'Pedro']]
        // Posicion:   0          1        2        3[0]      3[1]     3[2]

console.log(estudiantes[0]) // Martin
console.log(estudiantes[1]) // Fernando
console.log(estudiantes[2]) // Sara
console.log(estudiantes[3]) // Arreglo
console.log(estudiantes[3][0]) // Samuel
console.log(estudiantes[3][1]) // Jorge
console.log(estudiantes[3][2]) // Pedro
```

Asignación

Podemos asignar y reasignar cualquier índice en la matriz usando el paréntesis/índice y un "=".

```
var estudiantes = ['Martin', 'Fernando', 'Sara', ['Samuel', 'Jorge', 'Pedro']];
        // Posicion:   0          1        2        3[0]      3[1]     3[2]

estudiantes[0] = 'Juan';

console.log(estudiantes)
```

Métodos push() y pop()

Otros dos métodos de matriz incorporados muy útiles son .push y .pop. Estos métodos se refieren a la adición y eliminación de elementos de la matriz después de su declaración inicial.

```
var estudiantes = ['Martin', 'Fernando', 'Sara', 'Samuel'];

estudiantes.push('Patricia');

console.log(estudiantes);

estudiantes.pop();

console.log(estudiantes);
```

Métodos shift() y .unshift()

.unshift y **.shift** son exactamente como .push y .pop, excepto que operan en el primer elemento de la matriz. .unshift(item) colocará un nuevo elemento en la primera posición de la matriz, y .shift() eliminará el primer elemento de la matriz.

```
var estudiantes = ['Martin', 'Fernando', 'Sara', 'Samuel'];

estudiantes.unshift('Leo');

console.log(estudiantes);

estudiantes.shift();

console.log(estudiantes);
```

Tipos de datos compuestos: Objetos

Un objeto es una colección de datos relacionados y/o funcionalidad (que generalmente consta de algunas variables y funciones, que se denominan propiedades y métodos cuando están dentro de objetos).

```
var miPrimerObjeto = {
    nombre: 'Sherlock Holmes',
    edad: 60,
    genero: 'Masculino',
    intereses: ['Violin', 'Boxeo'],

    saludo: function() {
        alert('Hola, soy ' + this.nombre + '.')
    }
}

console.log(miPrimerObjeto)
console.log(miPrimerObjeto.saludo())
```

Operadores lógicos

Los operadores lógicos se utilizan normalmente con valores booleanos (lógicos); cuando lo son, devuelven un valor booleano. Sin embargo, los operadores && y || en realidad devuelven el valor de uno de los operandos especificados, por lo que si estos operadores se utilizan con valores no booleanos, pueden devolver un valor no booleano. Los operadores lógicos se describen en la siguiente tabla.

Operador	Descripción	Ejemplo
&&	AND: Este nos retorna verdadero si los dos valores a comparar son verdaderos de lo contrario retorna false	var1&&var2
||	OR: Nos retorna verdadero si alguno de los dos valores a comparar es verdadero sino false	var1||var2
!	NOT: Niega el valor, si la variable es true la devuelve falso y a la inversa	!var1

Estructuras de control de flujo: If - else

Esta estructura nos sirve como filtro para que solo se ejecuten instrucciones si se cumple una determinada condición. De lo contrario se disparan otras instrucciones diferentes.

```
if (/* Condicion */) {
    // Si la condicion se cumple
    // pasa esto.
} else {
    // Sino pasa esto
}
```

```
var numero = 9

if (numero < 10) {
    return true
} else {
    return false
}
```

Estructuras de control de flujo: **Switch - Case**

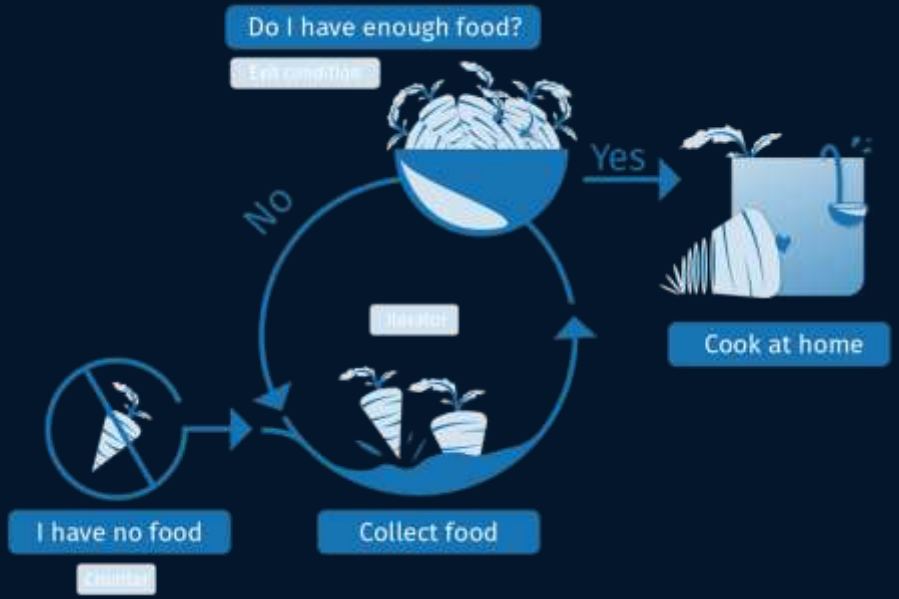
La declaración **switch** evalúa una expresión, comparando el valor de esa expresión con una instancia **case**, y ejecuta declaraciones asociadas a ese **case**, así como las declaraciones en los **case** que siguen.

```
var productos = 2;

switch (productos) {
    case 1:
        console.log("Té")
        break;
    case 2:
        console.log("Café")
        break;
    case 3:
        console.log("Agua")
        break;
    default:
        console.log("Caramelo media hora")
        break;
}
```

¿Qué es un bucle o ciclo?

Los bucles ofrecen una forma rápida y sencilla de hacer algo repetidamente. Hay muchos diferentes tipos de bucles, pero esencialmente, todos hacen lo mismo: repiten una acción varias veces. Los diversos mecanismos de bucle ofrecen diferentes formas de determinar los **puntos de inicio** y **terminación** del bucle. Hay varias situaciones que son fácilmente atendidas por un tipo de bucle que por otros.



Bucles

while

Crea un bucle que ejecuta una sentencia especificada mientras cierta condición se evalúe como verdadera. Dicha condición es evaluada antes de ejecutar la sentencia.

```
var contador = 0;

while (contador < 10) {
    console.log(contador);
    contador++
}
```

do while

La sentencia (hacer mientras) crea un bucle que ejecuta una sentencia especificada, hasta que la condición de comprobación se evalúa como falsa. La condición se evalúa después de ejecutar la sentencia, dando como resultado que la sentencia especificada se ejecute al menos una vez

```
var contador = 0;

do {
    console.log("do while" + contador);
    contador++
} while (contador < 10);
```

Bucles

for

Los bucles for tienen una sintaxis única, similar a la instrucción if, pero un poco más compleja. Primero tenemos la palabra clave for, seguida de paréntesis y luego abrir y llaves. Dentro de los paréntesis necesitaremos tres cosas. Primero, debemos declarar una variable, esto es sobre lo que se repetirá el bucle. Entonces tendremos una expresión condicional, el ciclo continuará sucediendo hasta que esta declaración sea false. Tercero, incrementaremos nuestra variable. Las tres declaraciones están separadas por un punto y coma.

```
for (var i = 0                ; i < 10                        ; i++) {
//  Declaramos una variable / Definimos una condicion / Incrementamos la variable
    console.log(i);
}
```

Bucles

for: iterando arrays

El Ciclo for también sirve para iterar sobre arreglos. Para esto debemos establecer una condición en la cual la variable sea menor a la longitud (length) del arreglo, de modo que luego de recorrer todo el arreglo el ciclo se corte. Lo veremos más claro en este ejemplo:

```
var arreglo = [10, 20, 30, 40, 50, 60, 70, 80, 90]

for (var i = 0; i < array.length; i++) {
    console.log(array[i]);
}
```

Manejo de errores: Try - Catch - Finally

La declaración **try...catch** señala un bloque de instrucciones a intentar (**try**), y especifica una respuesta si se produce una excepción (**catch**).

La sentencia **try** consiste en un bloque **try** que contiene una o más sentencias. Las llaves **{ }** se deben utilizar siempre, incluso para una bloques de una sola sentencia. Al menos un bloque **catch** o un bloque **finally** debe estar presente.

```
try {
    // Codigo a evaluar
} catch (error) {
    // Error a capturar
} finally {
    // Se ejecuta de todos modos
}
```

```
try {
    let numero = "z"

    if(isNaN(numero)) {
        throw new Error('El valor ingresado no es un numero');
    }

    console.log(numero * numero)
} catch (error) {
    console.log('Se produjo el siguiente error: ' + error)
}
```

ACADEMY
by NUMEN