JS SEMINARIO DE LENGUAJES

EXPLICACIÓN PRÁCTICA 2 (PARTE 1)

AVANZANDO EN JAVASCRIPT: OPERADORES, FUNCIONES Y MANIPULACIÓN DEL DOM

AGENDA

1. Operadores En Javascript
2. Tipos De Datos: Wrappers 1
3. Estructuras De Datos
4. Estructuras De Control

OPERADORES EN JAVASCRIPT

Los operadores nos permiten realizar acciones sobre variables y valores.

Existen diferentes tipos de operadores en JavaScript: aritméticos, de asignación, de comparación, y lógicos.

OPERADORES ARITMÉTICOS

Permiten realizar operaciones matemáticas básicas:

- `+` Suma
- `-` Resta
- `*` Multiplicación
- `/` División
- `%` Módulo (resto de la división)

```
let resultado = 10 + 5
```

OPERADORES DE INCREMENTO Y DECREMENTO

Aumentan o disminuyen el valor de una variable en 1:

- `++variable` Incrementa y retorna el valor.
- `variable++` Retorna el valor e incrementa el valor.
- `--variable` Decrementa y retorna el valor.
- `variable--` Retorna el valor y decrementa.

```
let a = 1
console.log(++a) // 2

let b = 1
console.log(b++) // 1
```

OPERADORES DE ASIGNACIÓN

Se utilizan para asignar valores a variables:

- `=` Asignación simple
- `+=` Suma y asigna
- `-=` Resta y asigna
- `*=` Multiplica y asigna
- `/=` Divide y asigna

```
x += 5 // Es equivalente a x = x + 5
```

OPERADORES DE COMPARACIÓN [1]

Comparan valores y retornan un booleano (`true` o `false`):

- `==`: Igualdad simple.
- `===`: Igualdad estricta
- `!=`: Desigualdad simple
- `!==`: Desigualdad estricta

```
console.log(5 <= '5') // true (igualdad simple)
console.log(5 === '5') // false (igualdad estricta)
```

Es importante usar `===` y `!==` para evitar errores.

OPERADORES DE COMPARACIÓN [2]

- `<`: menor que
- `<=`: menor o igual que
- `>`: mayor que
- `>=`: mayor o igual que

```
console.log(5 == '5') // true
console.log(5 <= '5') // true
console.log(6 <= '6') // false
```

Estos comparadores no validan el tipo.

OPERADORES LÓGICOS

Permiten combinar valores booleanos y determinar la lógica entre variables o valores.

- `&&`: AND
- `||`: OR
- `!`: NOT

```
if (edad >= 18 && nacionalidad == 'argentino') { // Código }
```

TIPOS DE DATOS: WRAPPERS [1]

En JavaScript se pueder llamar a métodos a cualquier tipo de dato.

Para el desarrollador, cualquier tipo de dato funcionaría coo un objeto, aunque sean tipos primitivos.

Esto se debe a que JavaScript envuelve los tipos primitivos en objetos (wrappers).

TIPOS DE DATOS: WRAPPERS [2]

Proporciona una manera de aplicar métodos y propiedades a tipos primitivos.

- Para operaciones avanzadas con cadenas de texto.
- para métodos numéricos adicionales.
- Boolean: para operaciones lógicas extendidas.

CONSIDERACIONES DE WRAPPERS

- **Autoboxing**: JavaScript convierte automáticamente los primitivos en objetos envoltorios cuando se requiere.
- **Precaución**: Crear objetos envoltorios manualmente (por ejemplo, usando new String, new Number, new Boolean) no es recomendado fuera de casos muy específicos debido a confusión y problemas de rendimiento.

ESTRUCTURAS DE DATOS

ARREGLOS

Un arreglo (array) es una lista ordenada de valores.

Cada valor en un array tiene un índice, comenzando desde 0.

```
let jedis = ['Obi-Wan Kenobi', 'Yoda', 'Luke Skywalker']
console.log(jedis[0]) // Obi-Wan Kenobi
```

ARREGLOS: CARACTERÍSTICAS

En JavaScript, los arreglos son:

- **Dinámicos**: pueden cambiar de tamaño dinámicamente. Se pueden agregar o quitar elementos en cualquier momento.
- **Heterogéneos**: pueden contener elementos de diferentes tipos en el mismo arreglo, incluso otros arrays u objetos.

```
let jedis = ['manzana', 2, {a: 1, b: 2}, [1, 2, 3]]
console.log(frutas[0]) // manzana
```

ARREGLOS: MÉTODOS [1]

JavaScript proporciona un conjunto de métodos para agregar y eliminar elementos:

- `.push(elemento)`: agrega un elemento al final.
- `.pop()`: elimina el último elemento y lo retorna.
- `.unshift(elemento)`: agrega un elemento al principio.
- `.shift()`: elimina el primer elemento y lo retorna.

ARREGLOS: MÉTODOS 2

```
let siths = ['Darth Plagueis', 'Palpatine']
siths.push('Darth Vader')
console.log(siths) // ['Darth Plagueis', 'Palpatine', 'Darth Vader']

const lastSith = siths.pop()
console.log(lastSith) // 'Darth Vader'

const firstSith = siths.shift()
console.log(firstSith) // 'Darth Plagueis'

siths.unshift('Darth Bane')
console.log(siths) // ['Darth Bane', 'Palpatine']
```

ARREGLOS: MÉTODOS [3]

Además proporciona métodos que operan sobre toda la estructura:

- `.filter(function)`: retorna un arreglo de elementos filtrados.
- `.map(function)`: retorna un arreglo con cada elemento operado.
- `.forEach(function)`: ejecutar una función por cada elemento.

ARREGLOS: MÉTODOS 4

```
let ages = [18, 25, 7, 30, 40, 5]
const adults = ages.filter(age => age >= 18)
console.log(adults) // [18, 25, 30, 40]
console.log(ages) // [18, 25, 7, 30, 40, 5]

const doubledAges = ages.map(age => age * 2)
console.log(doubledAges) // [36, 50, 14, 60, 80, 10]
console.log(ages) // [18, 25, 7, 30, 40, 5]

ages.forEach(age => console.log(age))
```

OBJETOS

Un objeto es una colección de propiedades, y una propiedad es una asociación entre un nombre (o clave) y un valor.

Un valor de propiedad puede ser una función, en cuyo caso la propiedad es conocida como un método.

```
let person = {
  name: 'Anakin Skywalker',
  age: 45,
  jedi: true,
  showName: function() { console.log(this.name) }
}
console.log(person.name) // Anakin Skywalker
person.showName() // Anakin Skywalker
```

OBJETOS: CARACTERÍSTICAS

En JavaScript, los objetos son:

- **Dinámicos**: las propiedades pueden agregarse y eliminarse después de la creación del objeto.
- **Flexibles**: pueden contener diferentes tipos de datos como números, strings, funciones y otros objetos.

```
const jedi = {
  nombre: 'Anakin Skywalker',
}
jedi.titulo = 'Darth Vader' // Agrega una nueva propiedad
console.log(jedi) // { nombre: 'Anakin Skywalker', titulo: 'Darth Va
```

OBJETOS: MÉTODOS BÁSICOS

JavaScript proporciona métodos para manipular objetos:

- `Object.keys(objeto)`: retorna un array de las claves del objeto.
- `Object.values(objeto)`: retorna un array de los valores de las propiedades del objeto.

OBJETOS: MÉTODOS AVANZADOS

```
const sith = {
  nombre: 'Palpatine',
  titulo: 'Emperador',
}
console.log(Object.keys(sith)) // ['nombre', 'titulo']
console.log(Object.values(sith)) // ['Palpatine', 'Emperador']

// Usando Object.entries para obtener clave-valor como arrays
Object.entries(sith).forEach(([clave, valor]) => {
  console.log(`${clave}: ${valor}`)
})
```

OBJETOS: MÉTODOS DE ITERACIÓN

Trabajar con objetos en loops y funciones de orden superior:

- `.forEach()` (usando `Object.entries`): ejecutar una función por cada par clave-valor.

ESTRUCTURAS DE CONTROL

Tenemos dos grupos de estructuras de control:

ESTRUCTURAS DE CONTROL

Tenemos dos grupos de estructuras de control:

- Estructuras condicionales

ESTRUCTURAS DE CONTROL

Tenemos dos grupos de estructuras de control:

- Estructuras condicionales
- Estructuras iterativas

ESTRUCTURAS CONDICIONALES

ESTRUCTURAS CONDICIONALES

- `if`: Condicional simple.

ESTRUCTURAS CONDICIONALES

- `if`: Condicional simple.
- `if/else`: Condicional con alternativa.

ESTRUCTURAS CONDICIONALES

- `if`: Condicional simple.
- `if/else`: Condicional con alternativa.
- operador ternario: Condicional en una sola línea.

ESTRUCTURAS CONDICIONALES

- `if`: Condicional simple.
- `if/else`: Condicional con alternativa.
- operador ternario: Condicional en una sola línea.
- `switch`: Múltiples condicionales.

ESTRUCTURAS CONDICIONALES: IF

ESTRUCTURAS CONDICIONALES: IF

```
if (condition) {
  // Código si la condición es verdadera
}
```

ESTRUCTURAS CONDICIONALES: IF/ELSE

ESTRUCTURAS CONDICIONALES: IF/ELSE

```
if (condition) {
  // Código si la condición es verdadera
} else {
  // Código si la condición es falsa
}
```

ESTRUCTURAS CONDICIONALES: OPERADOR TERNARIO

ESTRUCTURAS CONDICIONALES: OPERADOR TERNARIO

```
const valor = (condition) ? 'verdadero' : 'falso'
```

ESTRUCTURAS CONDICIONALES: SWITCH

ESTRUCTURAS CONDICIONALES: SWITCH

```
switch (expression) {
  case value1:
    // Código
    break
  case value2:
    // Código
    break
  default:
    // Código
}
```

ESTRUCTURAS ITERATIVAS

ESTRUCTURAS ITERATIVAS

- `while`: Iteración con una condición.

ESTRUCTURAS ITERATIVAS

- `while`: Iteración con una condición.
- `for`: Iteración con un contador.

ESTRUCTURAS ITERATIVAS

- `while`: Iteración con una condición.
- `for`: Iteración con un contador.
- `do/while`: Iteración con una condición al final.

ESTRUCTURAS ITERATIVAS

- `while`: Iteración con una condición.
- `for`: Iteración con un contador.
- `do/while`: Iteración con una condición al final.
- `for .. in`: Iterar sobre elementos.

ESTRUCTURAS ITERATIVAS

- `while`: Iteración con una condición.
- `for`: Iteración con un contador.
- `do/while`: Iteración con una condición al final.
- `for .. in`: Iterar sobre elementos.
- `for .. of`: Iterar sobre valores de un objeto.

ESTRUCTURAS ITERATIVAS: WHILE

ESTRUCTURAS ITERATIVAS: WHILE

```
while (condition) {
  // Código
}
```

ESTRUCTURAS ITERATIVAS: FOR

ESTRUCTURAS ITERATIVAS: FOR

```
for (let i = 0; i < 10; i++) {
  // Código
}
```

ESTRUCTURAS ITERATIVAS: DO/WHILE

ESTRUCTURAS ITERATIVAS: DO/WHILE

```
do {
  // Código
} while (condition)
```

ESTRUCTURAS ITERATIVAS: FOR .. IN

ESTRUCTURAS ITERATIVAS: FOR .. IN

```
for (let key in object) {
  // Código
}
```

ESTRUCTURAS ITERATIVAS: FOR .. OF

ESTRUCTURAS ITERATIVAS: FOR .. OF

```
for (let value of object) {
  // Código
}
```

FIN DE LA PARTE 1