REPASO DE JAVASCRIPT

TC3005B. Desarrollo e implantación de sistemas de software

Repaso

- let, const, var
- Arrow functions
- export, import
- Operador spread, rest (...)
- Operador de desestructuración.
- Funciones de arreglos. (filter, map)

M. en C. Roberto Martinez Roman - rmroman@tec.mx

var, let, const

- Las variables declaradas con **var** se pueden volver a declarar y modificar :(

```
var a = "Hola";
console.log(a);
var a = 10;
console.log(a)
```

- Si se declaran fuera de una función, son visibles de manera global.
- Si se declaran dentro de una función su ámbito es la función.

```
function saludar() {

    console.log("Hola mundo");
    if ( 1===1) {
        var b = 34;
    }

    console.log(b);
}
```


Problemas con var

```
function saludar() {
    var b = 500;
    if ( 1===1) {
        var b = 34;
    }
    console.log(b);
}
```

M. en C. Roberto Martinez Roman - rmroman@tec.mx

var, let, const

- **let** permite definir una variable cuyo ámbito es el bloque donde se declara.

```
function saludar() {
    let b = 500;
    if ( 1===1) {
        let b = 34;
    }
    console.log(b);
}
```


var, let, const

- **const** declara variables de solo lectura, es decir, constantes.
- Tienen un ámbito de bloque, igual que let.
- Las variables declaradas con const no pueden modificarse ni volver a declararse.

Arrow functions

- Una función flecha se escribe así:

➡
```
let func = (arg1, arg2, ..., argN) => expression;
```

➡ Esto crea una función que acepta N argumentos, evalúa la expresión después de => y regresa el resultado como valor de la función.

- Es equivalente a:

➡
```
let func = function(arg1, arg2, ..., argN) {
  return expression;
};
```


Arrow functions

```
let sum = (a, b) => {
  let result = a + b;
  return result;
};
```

export, import

- **export** permite exportar elementos de un archivo fuente.
- **import** permite importar elementos externos (previamente exportados) a un archivo fuente.

Operador spread, rest (...)

- El operador **spread** expande un arreglo/string en sus elementos individuales.

```
const numbers = [1, 2, 3];

console.log(...numbers);
console.log(numbers)
```

- El operador **rest** condensa en un solo elemento un conjunto de valores.

```
function sum(...theArgs) {
  let total = 0;
  for (const arg of theArgs) {
    total += arg;
  }
  return total;
}

console.log(sum(1, 2, 3));
// Expected output: 6

console.log(sum(1, 2, 3, 4));
// Expected output: 10
```

Operador de desestructuración

- Permite leer solo ciertos valores de una colección. Normalmente para asignarlos.

```
let a, b, rest;
[a, b] = [10, 20];

console.log(a);
// Expected output: 10

console.log(b);
// Expected output: 20

[a, b, ...rest] = [10, 20, 30, 40, 50];

console.log(rest);
// Expected output: Array [30, 40, 50]
```


Funciones de arreglos

- map
- filter
- sort

map

- Crea un nuevo arreglo iterando sobre cada elemento al cual se le aplica una función.

```
const array1 = [1, 4, 9, 16];
const map1 = array1.map(x => x * 2);
console.log(map1);
```


filter

- Crea una copia *shallow* de una porción de un arreglo. Filtra los elementos que cumplen la prueba implementada por una función que se pasa como parámetro.

```
const words = ['spray', 'limit', 'elite', 'exuberant', 'destruction', 'present'];
const result = words.filter(word => word.length > 6);
console.log(result);
```

sort

- Ordena los elementos de un arreglo sobre él mismo.
- Por default el ordenamiento es ascendente y lo hace convirtiendo cada valor a cadena y comparando las secuencias UTF-16.

```
const months = ['March', 'Jan', 'Feb', 'Dec'];
months.sort();
console.log(months);

const array1 = [1, 30, 4, 21, 100000];
array1.sort();
console.log(array1);
```

```
> Array ["Dec", "Feb", "Jan", "March"]
> Array [1, 100000, 21, 30, 4]
```

```
» arr1.sort( (a,b) => a>b );
← ▸ Array(5) [ 1, 4, 21, 30, 10000 ]
```

M. en C. Roberto Martinez Roman - rmroman@tec.mx