


Arrays / flat()


Se utiliza para aplanar un arreglo multidimensional, es decir, convierte un arreglo que contiene otros arreglos anidados en un solo arreglo con todos los elementos en el mismo nivel. Por ejemplo, si tenemos un arreglo que contiene varios arreglos anidados, `flat()` nos permite obtener un solo arreglo que contiene todos los elementos de los arreglos internos en el mismo nivel que los elementos del arreglo principal. Esto simplifica la manipulación y el procesamiento de datos en arreglos complejos.

js

 Copiar código

```
// Usando flat() - Unifica
const numeros = [1,2,3,5,6,[7,8,9,10,[12,45,75]]];
console.log(numeros.flat());
```

js

 Copiar código

```
// ejemplo 2
const numeros2 = [1,2,3,5,6,[7,8,9,10,[12,45,75]]];
console.log(numeros2.flat(2));
```


Arrays / isArray()

Se utiliza para verificar si un valor dado es un arreglo o no. Retorna `true` si el valor es un arreglo y `false` si no lo es. Esto es útil cuando se necesita determinar si un objeto es un arreglo antes de aplicar operaciones específicas de arreglos, como iterar sobre sus elementos o aplicar métodos

de arreglos. Por ejemplo, al utilizar `isArray()`, se puede evitar errores al intentar acceder a métodos de arreglos en valores que no son arreglos.

07.01.2024 MOD-2

js


 Copiar código

```
var arr1 = [1,2,3,4,5,6,7,8,9,10];  
console.log(Array.isArray(arr1));
```

Arrays / from()

Se utiliza para crear un nuevo arreglo a partir de un objeto iterable o de un objeto similar a un arreglo. Este método es útil cuando se necesita convertir objetos iterables, como cadenas de texto o arreglos similares, en arreglos reales. Por ejemplo, `Array.from()` puede ser usado para convertir una cadena de texto en un arreglo de caracteres o para copiar los elementos de un arreglo a otro arreglo nuevo. Además, `from()` permite especificar una función de mapeo para transformar los elementos durante la creación del nuevo arreglo.

js

 Copiar código


```
console.log(Array.from("2345456"));
```

Operador Rest(...)

El operador de propagación REST en JavaScript, representado por tres puntos (...), se utiliza para recoger argumentos indefinidos en funciones o para desestructurar arreglos y objetos. En el

contexto de las funciones, REST permite capturar un número variable de argumentos en un solo parámetro, convirtiéndolos en un arreglo. Esto proporciona flexibilidad al definir funciones que pueden manejar cualquier cantidad de argumentos sin necesidad de especificarlos individualmente. Por otro lado, en la desestructuración de arreglos y objetos, REST permite extraer elementos o propiedades restantes después de capturar las que se desean, simplificando la manipulación de datos estructurados. En resumen, REST es una característica poderosa de JavaScript que facilita la escritura de código más conciso y flexible.


js

 Copiar código

```
function sumar (a,b,c, ...otros){
  let suma = a + b + c;
  otros.forEach(n => suma += n);
  return suma;
}

console.log(sumar(2,4,5,5));
console.log(sumar(2,4,5,5,3));
console.log(sumar(2,4,5,5,3,4,5));
console.log(sumar(2,4,5,5,5,8,9));
console.log(sumar(2,4,5,5));
```

js

 Copiar código

```
// Ejemplo 2
function myFun(a,b, ...otros){
  console.log("a", a);
  console.log("b", b);
  console.log("manyMoreArgs", otros);
}


myFun("one", "two", "Three", "For", "Five");
```

Operador Spread

El operador Spread en JavaScript, representado por tres puntos (...), se utiliza para descomponer arreglos o objetos en elementos individuales. En el contexto de arreglos, el operador Spread

puede copiar elementos de un arreglo existente en uno nuevo o combinar múltiples arreglos en uno solo. En cuanto a los objetos, el operador Spread permite combinar múltiples objetos en uno solo, sobrescribiendo las propiedades en caso de colisión. Esta funcionalidad es especialmente útil para pasar argumentos a funciones, clonar objetos y crear nuevas estructuras de datos de manera rápida y sencilla. En general, el operador Spread es una herramienta versátil que simplifica la manipulación y transformación de datos

js

 Copiar código

```
// Operador Spread
const nu = [1,2,3,4,5,6];
const num = [7,8,9,10,11,12];
console.log(nu, num);


const arre1 = [...nu, ...num];
console.log(arre1);

const arre2 = [0, ...nu, "numero", ...num];
console.log(arre2);
```

Pilas

Las pilas son una estructura de datos que sigue el principio de LIFO (Last In, First Out), lo que significa que el último elemento agregado es el primero en ser eliminado. Se pueden implementar utilizando arreglos o listas enlazadas. Las operaciones básicas en una pila son push() para agregar un elemento, pop() para eliminar el último elemento agregado y peek() para obtener el valor del elemento superior sin eliminarlo. Las pilas son útiles para resolver problemas que requieren seguimiento de estado, como la reversión de cadenas, la evaluación de expresiones matemáticas y la navegación hacia atrás en aplicaciones web.

js

 Copiar código

```
// PILAS
class Stack {
  constructor(){
    this.stack = [];
  }
}
```

```

push(element){
  this.stack.push(element);
  return this.stack;
}

pop(){
  return this.stack.pop();
}

peek(){
  return this.stack[this.stack.length - 1];
}


size(){
  return this.stack.length;
}

print(){
  console.log(this.stack);
}
}

const stack = new Stack();
console.log(stack.size());
console.log(stack.push('Jhon Cena'));
console.log(stack.push('The Rock'));
console.log(stack.size());
stack.print();
console.log(stack.peek());
console.log(stack.pop());

```

js

 Copiar código

```

// Usando PILAS con NODOS
// Creamos una clase para cada nodo dentro de la pila.

class Node{
  // cada nodo va a tener dos propiedad, valor o apuntador(Nodo que sigue)
  constructor(value){
    this.value = value;
    this.siguiente = null;
  }
}

// Creamos la clase pila
class Stack{
  // La pila tiene tres propiedades, el primer nodo, el ultimo nodo y el tamaño de la pila
  constructor(){
    this.primeros = null;
    this.ultimo = null;
    this.size = 0;
  }
  // retornar el ultimo valor ingresado a la pila
  peek(){
    return this.primeros;
  }
  // creamos el método push el cual recibe un valor y lo ingresa a la pila
  push(val){
    let nNode = new Node(val);

```

```
    if(!this.primer0){
        this.primer0 = nNode;
        this.ultimo = nNode;
    }else{
        let temp = this.primer0;
        this.primer0 = nNode;
        this.primer0.siguiente = temp;
    }
    return ++ this.size;
}

// pop() elimina el elemento de la parte superior de la pila
pop(){
    if(!this.primer0) return null
    let tem = this.primer0;
    if(this.primer0 === this.ultimo){
        this.ultimo = null;
    }

    this.primer0 = this.primer0.siguiente;
    this.size--
    return tem.value
}

}
```

```
const stack = new Stack;
stack.push("dato1");
stack.push("dato2");
stack.push("dato3");
console.log(stack.primer0);
console.log(stack.peak());
console.log(stack.size);
console.log(stack.pop());
console.log(stack.peak());
console.log(stack);
```