ACTIVIDAD
JavaScript

Cesar Jesus Lara Avila

Calagua Mallqui Jairo Andre

20210279F

Facultad de Ciencias, Universidad Nacional de Ingeniería

noviembre del 2023

Comparaciones booleanas:

1. ‘undefined == null’: **VERDADERO**. Ambos son considerados iguales en comparaciones no estrictas (‘==’) porque son tipos de valor nulo.

2. ‘NaN == NaN’: **FALSO**. ‘NaN’ nunca es igual a otro ‘NaN’ en JavaScript. Podemos usar ‘isNaN()’ para verificar si un valor es ‘NaN’.

3. ‘null == false’: **FALSO**. ‘null’ no es igual a ‘false’ en JavaScript

4. ‘o == false’: **VERDADERO**. En contextos booleanos, tano ‘o’ como ‘false’ se consideran equivalentes.

5. ‘ “ ” == false’: **VERDADERO**. En contextos booleanos, una cadena vacía ‘ “” ’ se evalúa como ‘false’.

Comportamientos extraños de arrays:

1. ‘[1, 2, 3] + [4, 5, 6]’: Esto no produce la concatenación esperada de arrays. En JavaScript, al sumar dos arrays con el operador ‘+’, se convierten a cadenas y se concatenan, dando como resultado ‘”1, 2, 34, 5, 6””
2. ‘!![]’: Se convierte a ‘true’. Un array en JavaScript, aunque esté vacío, es un objeto y, por lo tanto, se evalúa como verdadero en un contexto booleano.
3. ‘[] == true’: FALSO. Aunque ‘[]’ se evalúa como verdadero, no es igual a ‘true’ en una comparación de igualdad no estricta (‘==’)
4. ‘[10, 1, 3].sort()’: Ordena el array en orden lexicográfico (alfabético) por defecto, lo que resultaría en ‘[1, 10, 3]’.
5. ‘[] == 0’: VERDADERO. En una comparación no estricta, JavaScript intenta convertir ambos lados a un tipo común y en este caso ‘[]’ se convierte a ‘0’ durante la comparación.

Clausuras:

El código que se nos proporciona se crea una función ‘f1’ que toma un argumento ‘x’, inicializa ‘baz’ en ‘3’, y devuelve una función anónima. Cuando se llama a ‘bar(11)’, imprime ‘19’ (5 + 11 + 3) en la consola.

Algoritmos:

La función ‘greatesNumber’ actualmente tiene una complejidad de tiempo de O(N^2). Para mejorarla a O(N), podemos simplemente recorrer el array una vez, manteniendo una variable para rastrear el mayor número encontrado hasta ahora.

La función ‘containsX’ tiene una complejidad de tiempo de O(N), ya que recorre la cadena una vez para verificar si contiene la letra “X”. No hay una manera más eficiente de encontrar un carácter específico en una cadena que no sea recorrerla.

Función para encontrar el primer carácter no duplicado:

```javascript
function firstNonDuplicateChar(str) {
  const charCount = {};

  for (let i = 0; i < str.length; i++) {
    const char = str[i];
    charCount[char] = (charCount[char] || 0) + 1;
  }

  for (let i = 0; i < str.length; i++) {
    const char = str[i];
    if (charCount[char] === 1) {
      return char;
    }
  }

  return null; // Retornar null si no se encuentra un carácter no duplicado
}

// Ejemplo de uso:
const inputString = "charizard";
const firstNonDupChar = firstNonDuplicateChar(inputString);
console.log("El primer carácter no duplicado es:", firstNonDupChar);
```

Esta función recorre la cadena dos veces. La primera vez, crea un mapa de conteo de caracteres y la segunda vez, busca el primer carácter que tenga un conteo de uno en el mapa, lo que indica que no está duplicado. Si no se encuentra ninguno, devuelve ‘null’.

Clases:

Según lo que me piden con el diseño de 2 clases, una llamada “Pokemon” y otra llamada “Charizard” y lo que debe hacer cada una, planteo lo siguiente:

```
class Pokemon {
  constructor(HP, attack, defense) {
    this.HP = HP;
    this.attack = attack;
    this.defense = defense;
    this.move = "";
    this.level = 1;
    this.type = "";
  }

  flight() {
    throw new Error("No se ha especificado ningún movimiento.");
  }

  canFly() {
    if (!this.type) {
      throw new Error("No se ha especificado un tipo.");
    }
    return this.type.includes("volar");
  }
}

class Charizard extends Pokemon {
  constructor(HP, attack, defense, move) {
    super(HP, attack, defense);
    this.move = move;
    this.type = "fuego/volar";
  }

  fight() {
    if (this.move) {
      console.log(`Usando movimiento ${this.move}`);
      return this.attack;
    } else {
      throw new Error("Se requiere especificar un movimiento.");
    }
  }
}
```

Estas clases representan un esquema básico de Pokemon y Charizard, con métodos que reflejan sus capacidades y comportamientos.