CJAVA

CJAVA

siempre para apoyarte

Visión

Poder aportar al desarrollo del País usando tecnología Java.

CJAVA
siempre para apoyarte

Quienes Somos

Somos una organización orientada a desarrollar, capacitar e investigar tecnología JAVA a través de un prestigioso staff de profesionales a nivel nacional.

CJAVA
siempre para apoyarte

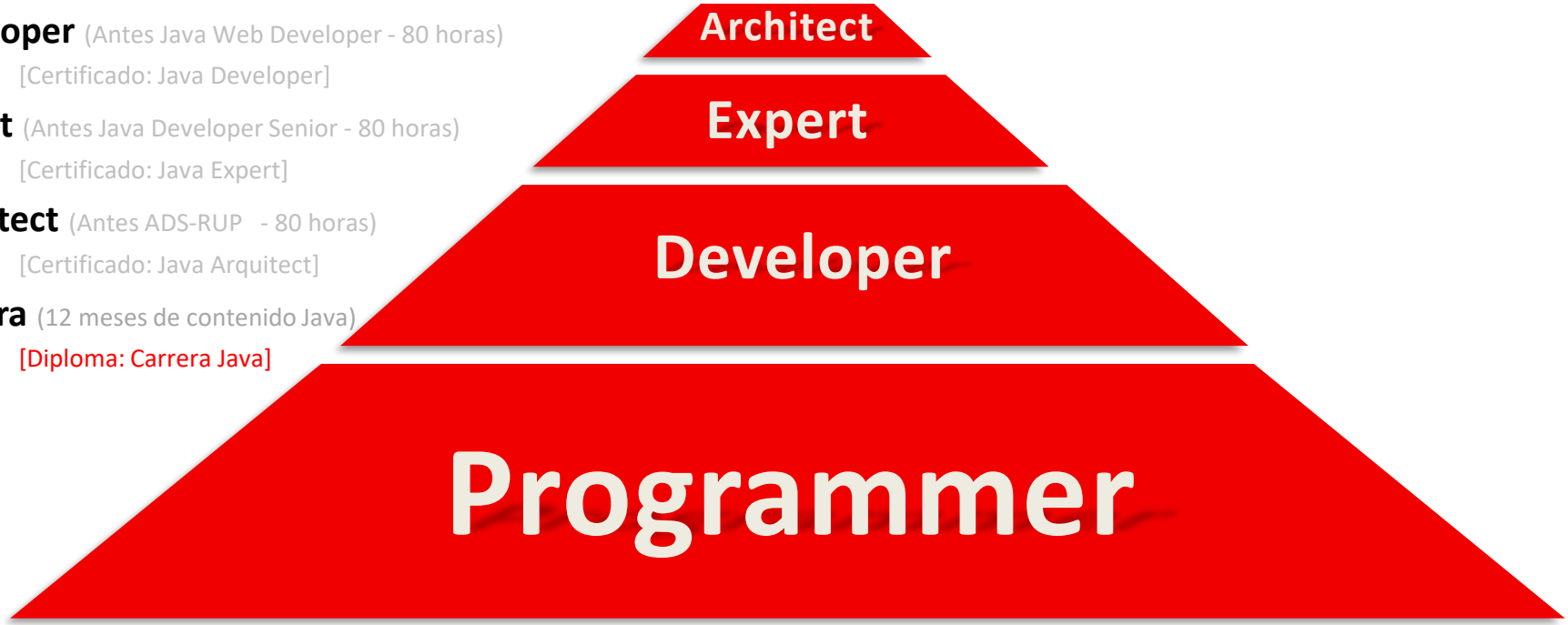
cjavaperu.com
info@cjavaperu.com

COMUNIDAD ACADEMICA

CJAVA
siempre para apoyarte

Servicio de Capacitación

- **Programer** (Antes Java Developer Junior - 80 horas)
  [Certificado: Java Programer]
- **Developer** (Antes Java Web Developer - 80 horas)
  [Certificado: Java Developer]
- **Expert** (Antes Java Developer Senior - 80 horas)
  [Certificado: Java Expert]
- **Arquitect** (Antes ADS-RUP - 80 horas)
  [Certificado: Java Arquitect]
- **Carrera** (12 meses de contenido Java)
  [Diploma: Carrera Java]

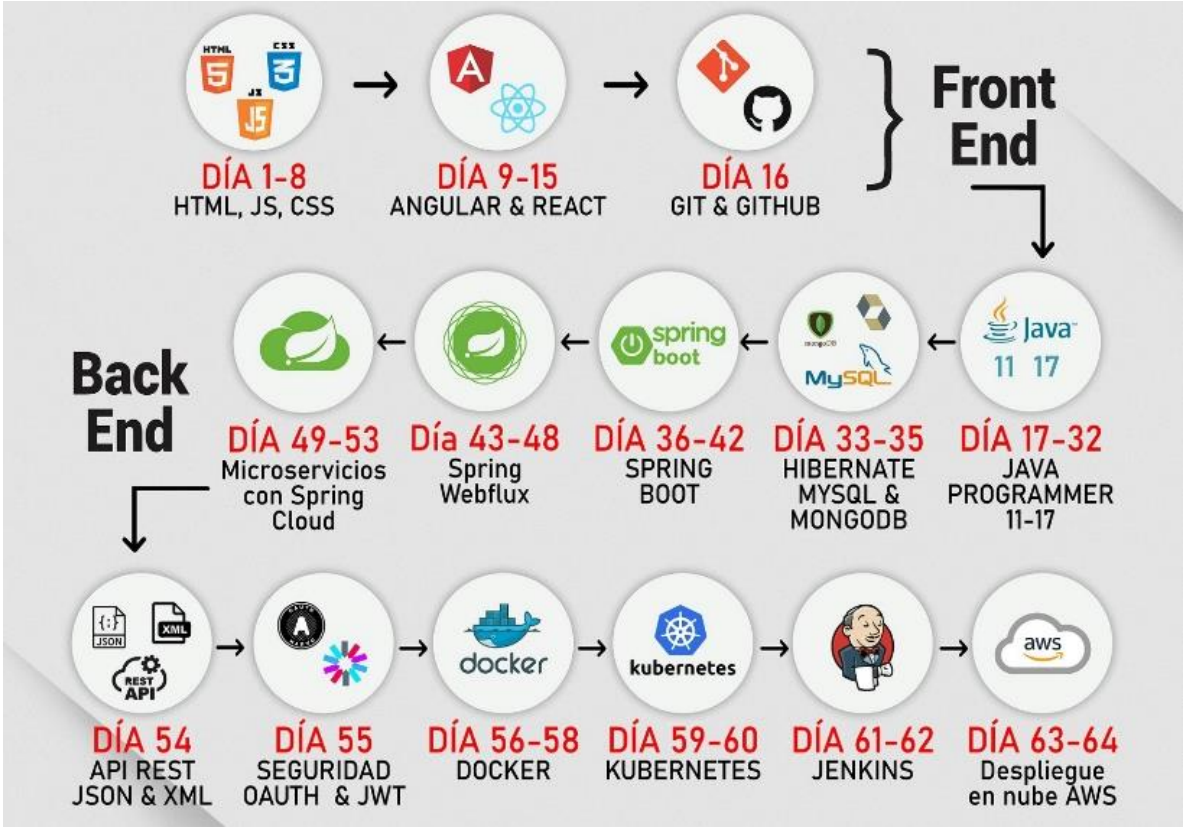


Java FullStack Developer

emaravi@cjavaperu.com

CJAVA
siempre para apoyarte

Introducción a Full Stack Development



Introducción a Full Stack Development

Front-end Development: Discutir las tecnologías y herramientas utilizadas en el desarrollo front-end, como HTML, CSS, JavaScript y React.

JavaScript

Edwin Maravi

CJAVA
siempre para apoyarte

Contenidos o temas

- Constructores en JavaScript
- Encapsulamiento en JavaScript
- Funciones get y set en JavaScript

CJAVA
siempre para apoyarte

Constructores en JavaScript

- El constructor, como su nombre indica, es el código que construye y configura un **objeto cuando se crea** con **new**. Todas las clases tienen un constructor, si no escribimos nada dentro de **class** se crea por defecto un constructor vacío. Si queremos escribir un constructor por nosotros mismos tenemos que incluir una función llamada constructor dentro del cuerpo de la clase.

CJAVA
siempre para apoyarte

Constructores en JavaScript

- El **método** constructor es un método especial para crear e inicializar un objeto creado a partir de una **clase**.
- **Sintaxis:**

```
constructor([argumentos]) { ... }
```

Constructores en JavaScript

- Sólo puede haber un método especial con el nombre de **"constructor"** en una clase. Un error de sintaxis será lanzado, si la clase contiene más de una ocurrencia de un método constructor.
- Un constructor puede utilizar la palabra clave **super** para llamar al constructor de una clase padre.
- Si no especifica un método constructor, se utiliza un **constructor predeterminado.**

Constructores en JavaScript

- **Ejemplo:**

```
class Square extends Polygon {
  constructor(length) {
    // Aquí, llama al constructor de la clase padre con sus longitudes
    // contemplando la anchura y la altura del Polígono
    super(length, length);
    // Nota: En las clases derivadas, super() se debe llamar primero
    // Se puede utilizar "this". Dejando esto causará un error de
    //referencia.
    this.name = 'Square';
  }

  get area() {
    return this.height * this.width;
  }

  set area(value) {
    this.area = value;
  }
}
```

- En JavaScript, la palabra clave **“this”** se refiere a un objeto.

Constructores en JavaScript

- Si no especifica un método constructor, se utiliza un constructor predeterminado. Para las clases base, el constructor por defecto es:

```
constructor() {}
```

- Para las clases derivadas, el constructor por defecto es:

```
constructor(...args) {
  super(...args);
}
```

CJAVA
siempre para apoyarte

Encapsulamiento en JavaScript

La encapsulación es el **empaquetamiento de datos** y funciones en un componente (por ejemplo, una clase) y para luego controlar el acceso a ese componente para hacer un efecto de "caja negra" fuera del objeto. Debido a esto, un usuario de esa clase solo necesita conocer su interfaz (es decir, los datos y las funciones expuestas fuera de la clase), no la implementación oculta.

CJAVA
siempre para apoyarte

get y set en JavaScript

- Una función que obtiene un valor de una propiedad se llama **getter** y una que establece el valor de una propiedad se llama **setter**.
- Esta característica a sido implementada en **ES2015,** pudiendo modificar el funcionamiento normal de establecer u obtener el valor de una propiedad, a estas se les conoce como accessor properties.

get y set en JavaScript

Las propiedades de acceso se construyen con métodos de obtención **“getter”** y asignación **“setter”.** En un objeto literal se denotan con **get y set:**

```
let obj = {
  get propName() {
    // getter, el código ejecutado para obtener obj.propName
  },

  set propName(value) {
    // setter, el código ejecutado para asignar obj.propName = value
  }
};
```

get y set en JavaScript

Ejemplo:

```
let user = {
  name: "John",
  surname: "Smith",

  get fullName() {
    return `${this.name} ${this.surname}`;
  },

  set fullName(value) {
    [this.name, this.surname] = value.split(" ");
  }
};

// set fullName se ejecuta con el valor dado.
user.fullName = "Alice Cooper";

alert(user.name); // Alice
alert(user.surname); // Cooper
```

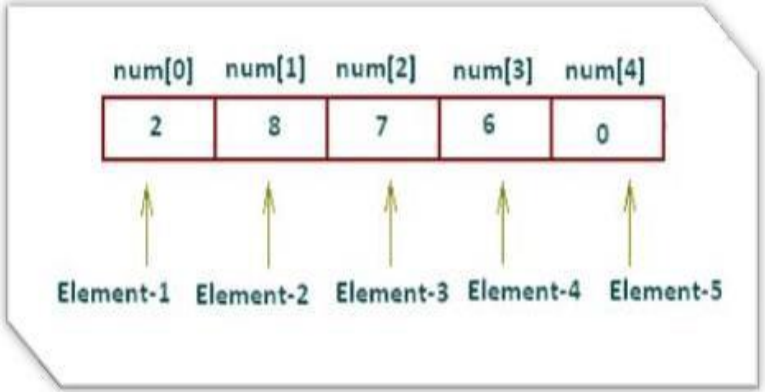
CJAVA
siempre para apoyarte

Contenidos o temas

- Arreglos en JavaScript
- Operaciones con arreglos
- Acceso a los elementos de un arreglo
- Recorridos de arreglos

Arreglos en JavaScript

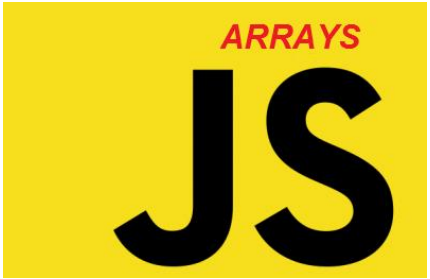
En programación, un arreglo es una **colección de elementos** o cosas. Los arreglos **guardan** datos como elementos y los regresan cuando los necesitas.



CJAVA
siempre para apoyarte

Arreglos en JavaScript

El objeto **Array** de JavaScript es un **objeto** global que es usado en la construcción de arrays, que son objetos **tipo lista** de alto nivel.



CJAVA
siempre para apoyarte

Arreglos en JavaScript

Los **arrays** 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. Dado que la longitud de un array puede cambiar en cualquier momento, y los datos se pueden almacenar en ubicaciones no contiguas, no hay garantía de que los arrays de JavaScript sean densos; esto depende de cómo el programador elija usarlos.

CJAVA
siempre para apoyarte

Operaciones con arreglos

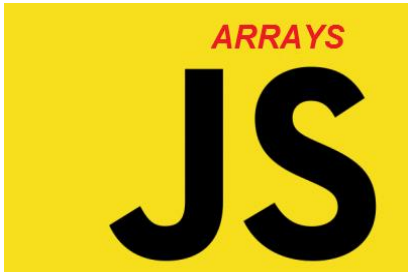
Los arrays cuentan con muchos métodos. Para hacer las cosas más sencillas. Algunos métodos que agregan o extraen elementos del inicio o final de un array:

arr.push(...items) – agrega ítems al final,

arr.pop() – extrae un ítem del final,

arr.shift() – extrae un ítem del inicio,

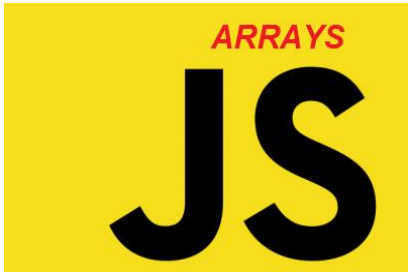
arr.unshift(...items) – agrega ítems al principio.



CJAVA
siempre para apoyarte

Operaciones con arreglos

Al igual que los String, saber el número elementos que tiene un array es muy sencillo. Sólo hay que acceder a la propiedad **.length**, que nos devolverá el número de elementos existentes en un array.



CJAVA
siempre para apoyarte

Operaciones con arreglos

Crear un array:

```
let frutas = ["Manzana", "Banana"]

console.log(frutas.length)
// 2
```

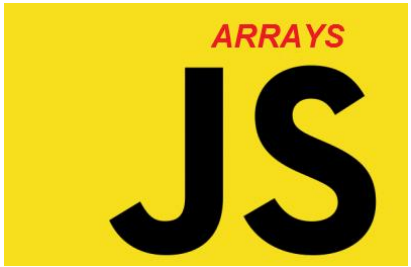
CJAVA
siempre para apoyarte

Operaciones con arreglos

Acceder a un elemento de Array mediante su índice:

```
let primero = frutas[0]
// Manzana

let ultimo = frutas[frutas.length - 1]
// Banana
```

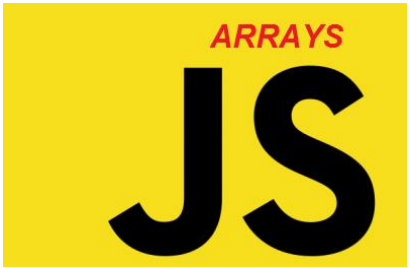


CJAVA
siempre para apoyarte

Operaciones con arreglos

Añadir un elemento al final de un Array:

```
let nuevaLongitud = frutas.push('Naranja') // Añade "Naranja" al final
// ["Manzana", "Banana", "Naranja"]
```

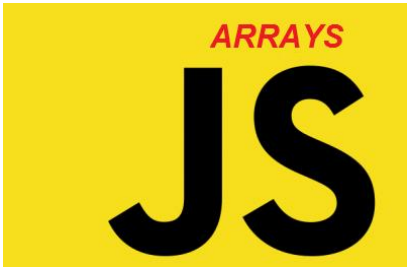


CJAVA
siempre para apoyarte

Operaciones con arreglos

Eliminar el último elemento de un Array:

```
let ultimo = frutas.pop() // Elimina "Naranja" del final
// ["Manzana", "Banana"]
```



CJAVA
siempre para apoyarte

Operaciones con arreglos

Añadir un elemento al principio de un Array:

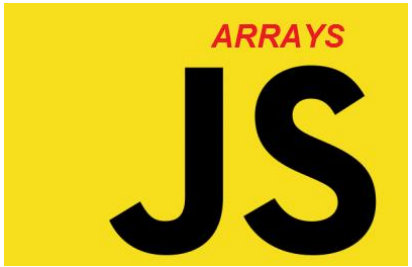
```
let nuevaLongitud = frutas.unshift('Fresa') // Añade "Fresa" al inicio
// ["Fresa" ,"Manzana", "Banana"]
```

CJAVA
siempre para apoyarte

Operaciones con arreglos

Eliminar el primer elemento de un Array:

```
let primero = frutas.shift() // Elimina "Fresa" del inicio
// ["Manzana", "Banana"]
```

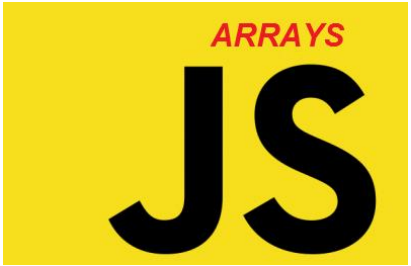


CJAVA
siempre para apoyarte

Acceso a los elementos de un arreglo

Los **índices** de los arrays de JavaScript comienzan en **cero**, es decir, el índice del primer elemento de un array es 0, y el del último elemento es igual al valor de la propiedad **length** del array restándole 1. Si se utiliza un número de índice no válido, se obtendrá **undefined**.

```
let arr = ['este es el primer elemento', 'este es el segundo elemento', 'este es el último elemento']
console.log(arr[0])              // escribe en consola 'este es el primer elemento'
console.log(arr[1])              // escribe en consola 'este es el segundo elemento'
console.log(arr[arr.length - 1]) // escribe en consola 'este es el último elemento'
```



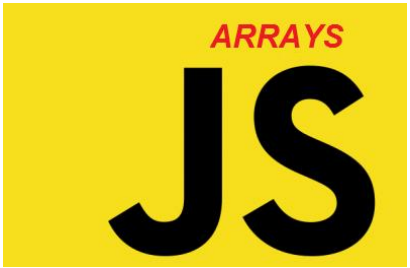
CJAVA
siempre para apoyarte

Recorrido de un arreglo

Recorrer un Array:

```
var array = [1, "Pedro", true, false, "Juan"];

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



CJAVA
siempre para apoyarte

Contenidos o temas

- Introducción a JSON
- Estructura de JSON
- JSON y Tipos de datos
- JSON y Arrays
- Conversiones de JSON

CJAVA
siempre para apoyarte

Introducción a JSON

JavaScript Object Notation (JSON) es un **formato** basado en texto estándar para representar **datos estructurados** en la sintaxis de **objetos** de JavaScript. Es comúnmente utilizado para **transmitir** datos en aplicaciones **web** (por ejemplo: enviar algunos datos desde el servidor al cliente, así estos datos pueden ser mostrados en páginas web, o vice versa).

CJAVA
siempre para apoyarte

Introducción a JSON

Los JSON son **cadenas** - útiles cuando se quiere **transmitir** datos a través de una **red**. Debe ser **convertido** a un objeto nativo de JavaScript cuando se requiera acceder a sus datos. Ésto no es un problema, dado que JavaScript posee un objeto global JSON que tiene los **métodos** disponibles para convertir entre ellos.

Un **objeto JSON** puede ser almacenado en su propio archivo, que es básicamente sólo un archivo de texto con una extension **.json**

Características de JSON

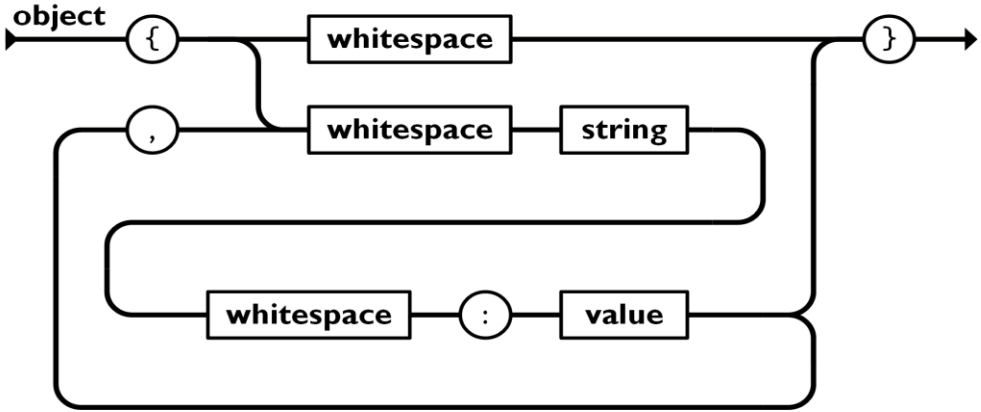
- JSON es un lenguaje de modelador de datos.
- Consiste en pares "clave - valor".
- Los valores pueden cadenas, números o boleanos, así como otros objetos JSON, con cualquier nivel de anidación.
- Es un formato flexible, ligero y fácilmente transferible a través de las redes.

Ventajas de JSON

- La lectura del código resulta de fácil lectura y la información es suficienteme expresiva para poder ser leída por personas, además de máquinas.
- El tamaño de los archivos que se transfieren es ligero.
- El código está basado en el lenguaje JavaScript, lo que es ideal para las aplicaciones web.
- Todos los lenguajes disponen de funciones para interpretar cadenas JSON y convertir datos en cadenas JSON válidas.
- Se escribe en archivos de texto plano con codificación UTF8, que es compatible con todos los sistemas.

Estructura de JSON

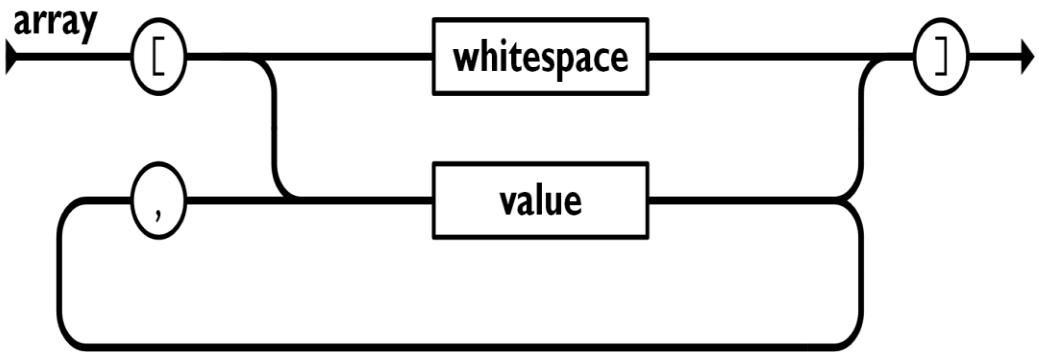
- En JSON, se presentan de **estas formas:**
  - Un **objeto** es un conjunto desordenado de pares nombre/valor. Un objeto comienza con **{ llave de apertura** y termine con **} llave de cierre.** Cada nombre es seguido por **: dos puntos** y los pares nombre/valor están separados por **, coma.**



Estructura de JSON

En JSON, se presentan de **estas formas:**

Un **arreglo** es una colección de valores. Un arreglo comienza con **[corchete izquierdo** y termina con **] corchete derecho.** Los valores se separan por **, coma.**

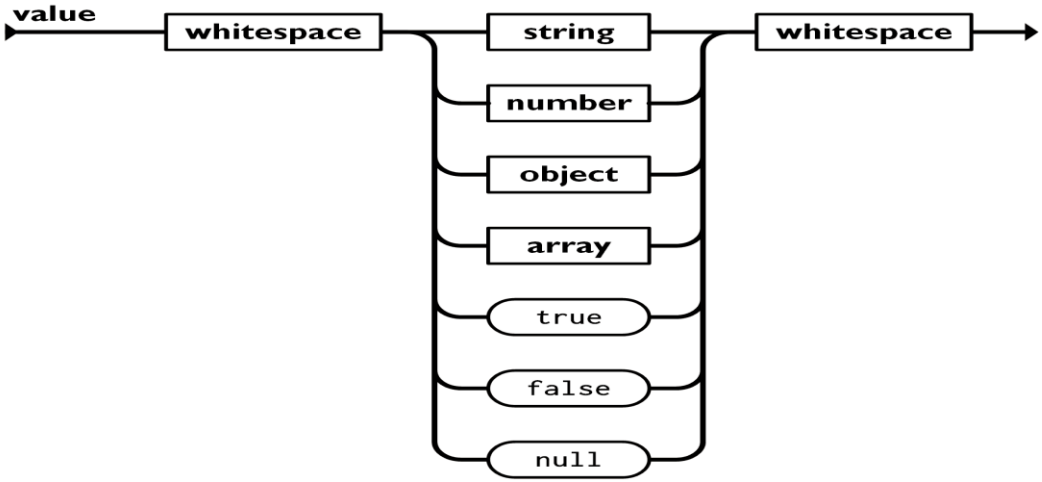


CJAVA
siempre para apoyarte

Estructura de JSON

En JSON, se presentan de **estas formas:**

Un **valor** puede ser una **cadena** de caracteres con comillas dobles, o un **número**, o **true o false** o **null**, o un **objeto** o un **arreglo**. Estas estructuras pueden **anidarse.**



CJAVA
siempre para apoyarte

Estructura de JSON

Ejemplo de JSON como **objeto**:

```
{
  "nombre":"Jonh Doe",
  "profesion":"Programador",
  "edad":25,
  "lenguajes":["PHP","Javascript","Dart"],
  "disponibilidadParaViajar":true,
  "rangoProfesional": {
      "aniosDeExperiencia": 12,
      "nivel": "Senior"
  }
}
```

Estructura de JSON

Ejemplo de JSON como **array**:

```
[
    {
        "id": 56431
        "name": "El Padrino",
        "year": 1972
    },
    {
        "id": 7553
        "name": "El Padrino 2",
        "year": 1974
    },
    {
        "id": 19563
        "name": "El Padrino III",
        "year": 1990
    },
]
```

Conversiones de JSON

En **Javascript** tenemos una serie de **métodos** que nos facilitan la tarea de pasar de JavaScript a JSON y viceversa, pudiendo trabajar con contenido de tipo (que contenga un JSON) y objetos JavaScript según nos interese.

MÉTODO	DESCRIPCIÓN
`JSON.parse(str)`	Convierte el texto (String) **str** (si es un JSON válido) a un objeto y lo devuelve.
`JSON.stringify(obj)`	Convierte un objeto Javascript (Object) **obj** a su representación JSON y la devuelve.
`JSON.stringify(obj, props)`	Idem al anterior, pero filtra y mantiene solo las propiedades del (Array) **props**.
`JSON.stringify(obj, props, spaces)`	Idem al anterior, pero indenta el JSON a (Number) **spaces** espacios.

Conversiones de JSON

Ejemplo con el mencionado método **JSON.parse():**

```
const json = ‘{
  "name": "Manz",
  "life": 99
}’;

const user = JSON.parse(json);

user.name;  // "Manz"
user.life;  // 99
```

Conversiones de JSON

Ejemplo con el mencionado método **JSON.stringify():**

```
const user = {
  name: "Manz",
  life: 99,
  talk: function () {
    return "Hola!";
  },
};

JSON.stringify(user);   // '{"name":"Manz","life":99}'
```

CJAVA

siempre para apoyarte

Gracias