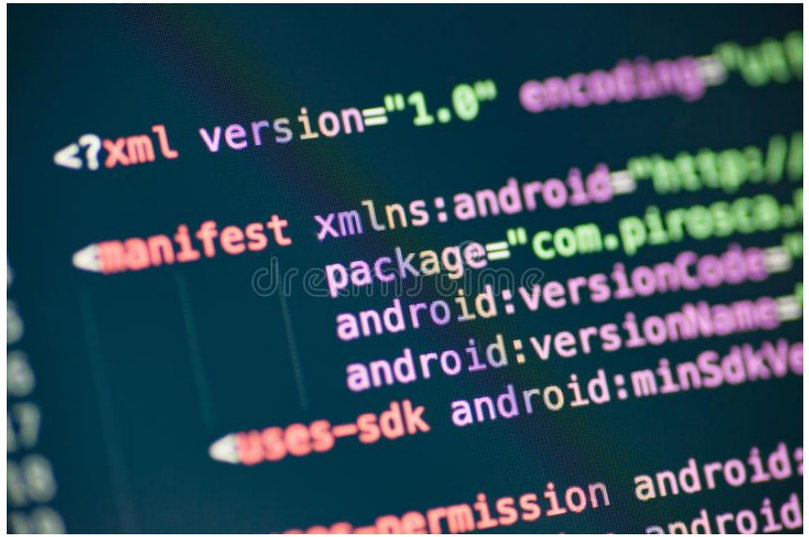
Lenguajes de marcas y sistemas de gestión de la información



UT04 – XML

3 – JSON

IES Clara del Rey – Madrid

JSON

JSON es el acrónimo de JavaScript Object Notation.

Es un subconjunto de la notación literal de objetos de JavaScript, pero que desde el año 2019 se considera independiente del lenguaje JavaScript.

No es un lenguaje de programación, es una notación texto para creación y definición de objetos, independiente de plataforma.

Se ha adoptado ampliamente como una alternativa a XML, fundamentalmente por la total integración del formato con JavaScript, y porque, en principio, es más fácil de procesar que XML (debatible).

Raíz de un documento JSON

Un documento JSON incluye una estructura jerárquica, que puede representar, en su nivel más bajo de la jerarquía, dos cosas diferentes:

- Un objeto, que se delimita con llaves : { ... }
- Una colección de valores / objetos, que se delimita con corchetes: [...]

Esta es una de las diferencias con XML, que siempre debe comenzar con un elemento raíz.

A partir de ahí, el documento JSON puede contener un anidamiento arbitrario de objetos y colecciones

Ejemplos

JSON que comienza con un objeto en la raíz del documento:

```json
{
    "dni": "23546365X",
    "nombre": "Antonio",
    "apellidos": "Carrasco Antunez",
    "fechaNacimiento": "1985-10-33"
}
```

JSON que comienza con una colección en la raíz del documento

```json
[
    { "name": "AliceBlue", "hexCode": "#F0F8FF" },
    { "name": "AntiqueWhite", "hexCode": "#FAEBD7" },
    { "name": "Aqua", "hexCode": "#00FFFF" }
]
```

Ejemplos

JSON con anidamiento de objetos y colecciones


```
[
    {
        "dni": "12345678D",
        "nombreApellidos": "Antonio Martínez Calvo",
        "calificaciones": [
            { "modulo": "SI", "nota": "4" }, { "modulo": "FOL", "nota": "5" },
            { "modulo": "PRO", "nota": "7" }, { "modulo": "LMM", "nota": "6" },
            { "modulo": "BD", "nota": "8" }, { "modulo": "ED", "nota": "9"}
        ],
        "aficiones": [ "Cocina", "Ajedrez", "Música", "Deporte" ]
    },
    {
        "dni": "87654321D",
        "nombreApellidos": "Martina Jimenez Salazar",
        "calificaciones": [
            { "modulo": "SI", "nota": "2" }, { "modulo": "FOL", "nota": "8" },
            { "modulo": "PRO", "nota": "4" }, { "modulo": "LMM", "nota": "2" },
            { "modulo": "BD", "nota": "7" }, { "modulo": "ED", "nota": "5"}
        ],
        "aficiones": [ "Series", "Amigos", "Fútbol", "Videojuegos" ]
    }
]
```

Características y usos de JSON

Características

- Fácil de leer y escribir, porque es autodescriptivo.
- Es ligero, basado en texto, fácil de comprimir al transmitirlo.
- Independiente del lenguaje.
- Soporte en todos los lenguajes modernos (no sólo JavaScript).

Usos

- Aplicaciones JavaScript, fundamentalmente para la comunicación de datos entre la aplicación y los servicios que consume.
- Comunicación de datos entre procesos o servicios de servidor.
- Ficheros de configuración de aplicaciones o servicios.

Sintaxis

Objetos

- Los objetos se delimitan con llaves {}.
- Las propiedades del objeto se representan con parejas nombre/valor.
- El nombre y el valor se separan por dos puntos (:) – *"prop": "...valor..."*
- El nombre de la propiedad siempre tiene que ir entre comillas dobles.
- Las propiedades se separan entre sí por comas.

Colecciones (arrays)

- Las colecciones de delimitan con corchetes []. Los elementos de la colección se separan con comas.
- Una colección puede ser de objetos, o de elementos primitivos como números, cadenas de caracteres, booleans.

Tipos de datos

Números:

- Enteros: 5, 7, 9, ...
- Decimales: con punto (.), no coma (,): 3.4546 ...
- Científica (potencias de 10): 0.3E-10, 5E+5, ...

Cadenas (string):

- Caracteres de escape: se usa \ para escapar la comilla y la barra invertida (\), igual que en Java.

Boolean:

- true / false (en minúscula)

Espacios en blanco y null

Se ignoran los espacios en blanco en todo el documento, salvo los que están dentro de los valores de las propiedades.

```
{
    "nombre":                "    estos espacios sí son relevantes        "
}
```

En este caso, los espacios amarillos se ignoran, los verdes no.

Se puede usar “null” para indicar propiedades nulas. En los arrays no es lo mismo [] (vacío) que “null” (nulo)

```
"nota": null,
"aficiones": []
```

JavaScript

Para incluir un fragmento de JavaScript en HTML se usa el tag "<script>".

Lo más ortodoxo es colocarlo en head, pero a veces se hace tras </body>.

```
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Prueba de JavaScript</title>
    <script>
        // Código JS
    </script>
</head>
```

Se puede enlazar un fichero de script externo, usando los atributos adecuados en el elemento script.

```
<title>Document</title>
<script src="script.js"></script>
```

JavaScript – Variables: var, const, let

Para definir una variable en JavaScript se puede:

- Usarla directamente, sin usar var, let o const.
- var: la forma “antigua” de definir variables. Se tiende a abandonar en favor de let y const, porque estas gestionan mejor el ámbito.
- let: declara una variable que puede ser modificada
- const: declara una variable que no puede cambiar de valor una vez asignado

Las variables en JS son de tipado dinámico. Reconoce ciertos tipos de datos (número, cadenas, fechas, ...) pero se puede reutilizar una variable para almacenar tipos diferentes.

JavaScript – Objetos

Los objetos se pueden declarar directamente usando JSON

```
const objeto = {
    nombrePropiedad1: valorPropiedad1,
    'nombrePropiedad2': valorPropiedad2 }
```

Se puede acceder a las propiedades de un objeto de distintas formas:

- `const valor = objeto.nombrePropiedad`
- `const valor = objeto['nombrePropiedad']`

Y se pueden añadir dinámicamente propiedades sin declararlas

- `objeto.nuevaPropiedad = valor`
- `objeto['nuevaPropiedad'] = valor`

JavaScript – Funciones

En JS las funciones pueden devolver o no valores. Todas se declaran como funciones independientemente de si devuelven o no valor.

Hay varias formas de declarar una función:

- Tradicional:

```
function nombreFuncion(param1, param2) { … }
```

- Como expresión:

```
const nombreFuncion = function (param1, param2) { … }
```

- Como expresión con lambdas
  - `const nombreFuncion = (param1, param2) => { … }`

JavaScript – Funciones

También pueden ser métodos de objetos:

```
const objeto = {
    prop1 : valorProp1,
    'prop2' : 'valorProp2',
    nombreMetodo1 : function (param) { … },
    nombreMetodo2 : (param) => { … }
}
```

Que se pueden invocar:

```
objeto.nombreMetodo1(parametro);
```