

# Referencia Rápida de JSON

Sintaxis, tipos de datos, objetos, arrays, jq

## Sintaxis

### Reglas

<code>{ }</code>	Objeto (pares clave-valor sin orden)
<code>[ ]</code>	Array (lista ordenada de valores)
<b>"key": value</b>	Las claves deben ser cadenas entre comillas dobles
<b>No trailing comma</b>	El último elemento no debe tener coma
<b>No comments</b>	JSON no permite comentarios

### Ejemplo mínimo

```
{
  "name": "Alice",
  "age": 30,
  "active": true
}
```

## Tipos de datos

### Seis tipos de valor

<b>"string"</b>	Texto UTF-8 entre comillas dobles
<b>42 / 3.14</b>	Número (entero o punto flotante)
<b>true / false</b>	Booleano
<b>null</b>	Nulo (ausencia de valor)
<b>{ }</b>	Objeto
<b>[ ]</b>	Array

### Secuencias de escape en cadenas

<code>\"</code>	Comilla doble
<code>\\</code>	Barra invertida
<code>\n \t</code>	Nueva línea, tabulador
<code>\uXXXX</code>	Escape Unicode (hexadecimal)

## Objetos

### Sintaxis de objeto

```
{
  "id": 1,
  "name": "Widget",
  "tags": ["new", "sale"]
}
```

### Reglas

<b>Keys</b>	Deben ser cadenas únicas entre comillas dobles
<b>Values</b>	Cualquier tipo JSON válido
<b>Order</b>	El orden de las claves no está garantizado
<b>Nesting</b>	Los objetos pueden contener objetos

## Arrays

### Sintaxis de array

```
[1, "two", true, null, {"key": "val"}]
```

### Array de tipos mixtos

```
{
  "matrix": [[1, 2], [3, 4]],
  "empty": []
}
```

### Reglas

<b>Ordered</b>	Los elementos mantienen el orden de inserción
<b>Mixed types</b>	Los elementos del array pueden ser de distintos tipos
<b>Indexing</b>	Base cero (en la mayoría de lenguajes)

## Anidamiento

### Estructura anidada

```
{
  "user": {
    "name": "Alice",
    "address": { "city": "Boston" },
    "scores": [95, 88, 72]
  }
}
```

### Patrones de acceso

<b>obj.user.name</b>	Notación de punto (JavaScript)
<b>obj["user"]["name"]</b>	Notación de corchetes
<b>obj.user.scores[0]</b>	Índice de array dentro de objeto anidado

## Validación de esquema

### Ejemplo de JSON Schema

```
{
  "type": "object",
  "properties": {
    "name": { "type": "string" },
    "age": { "type": "integer", "minimum": 0 }
  },
  "required": ["name"]
}
```

### Palabras clave del esquema

<b>type</b>	string, number, integer, boolean, object, array, null
<b>required</b>	Array de nombres de propiedades obligatorias
<b>properties</b>	Define las propiedades esperadas del objeto
<b>enum</b>	Restringir a un conjunto fijo de valores
<b>minLength / maxLength</b>	Restricciones de longitud de cadena
<b>minimum / maximum</b>	Restricciones de rango numérico

## Fundamentos de jq

### Filtros comunes

<b>.</b>	Identidad — pasar la entrada tal cual
<b>.key</b>	Acceder a la clave del objeto
<b>.key.nested</b>	Acceder a clave anidada
<b>.[0]</b>	Primer elemento del array
<b>.[ ]</b>	Iterar todos los elementos del array
<b>select(.age &gt; 20)</b>	Filtrar por condición
<b>map(.name)</b>	Transformar cada elemento
<b>length</b>	Longitud del array o cadena
<b>keys</b>	Claves del objeto como array

### Ejemplos de jq

```
echo '{"a":1}' | jq '.a' # 1
echo '[1,2,3]' | jq 'map(. * 2)' # [2,4,6]
cat data.json | jq '.users[].name'
cat data.json | jq '.[] | select(.active)'
```

## Patrones comunes

### Respuesta de API

```
{
  "status": 200,
  "data": [{"id": 1, "name": "Alice"}],
  "meta": {"total": 42, "page": 1}
}
```

## Archivo de configuración

```
{
  "host": "localhost",
  "port": 8080,
  "debug": false,
  "features": ["auth", "logging"]
}
```

## Consejos

<b>Validate</b>	Usar jsonlint o python -m json.tool
<b>Pretty print</b>	jq . file.json o python -m json.tool
<b>JSONL</b>	Un objeto JSON por línea (delimitado por salto de línea)
<b>JSON5 / JSONC</b>	Extensiones que permiten comentarios y comas finales