JSON Quick Reference

Syntax, data types, objects, arrays, jq

Syntax

Rules

`{ }`	Object (unordered key-value pairs)
`[ ]`	Array (ordered list of values)
`"key": value`	Keys must be double-quoted strings
`No trailing comma`	Last item must not have a comma
`No comments`	JSON does not allow comments

Minimal Example

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

Data Types

Six Value Types

`"string"`	Double-quoted UTF-8 text
`42 / 3.14`	Number (integer or floating point)
`true / false`	Boolean
`null`	Null (absence of value)
`{ }`	Object
`[ ]`	Array

String Escape Sequences

`\"`	Double quote
`\\`	Backslash
`\n  \t`	Newline, tab
`\uXXXX`	Unicode escape (hex)

Objects

Object Syntax

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

Rules

`Keys`	Must be unique double-quoted strings
`Values`	Any valid JSON type
`Order`	Key order is not guaranteed
`Nesting`	Objects can contain objects

Arrays

Array Syntax

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

Mixed Type Array

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

Rules

`Ordered`	Elements maintain insertion order
`Mixed types`	Array items can be different types
`Indexing`	Zero-based (in most languages)

Nesting

Nested Structure

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

Access Patterns

`obj.user.name`	Dot notation (JavaScript)
`obj["user"]["name"]`	Bracket notation
`obj.user.scores[0]`	Array index within nested object

Schema Validation

JSON Schema Example

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

Schema Keywords

`type`	string, number, integer, boolean, object, array, null
`required`	Array of required property names
`properties`	Defines expected object properties
`enum`	Restrict to a fixed set of values
`minLength / maxLength`	String length constraints
`minimum / maximum`	Number range constraints

jq Basics

Common Filters

`.`	Identity — pass input through
`.key`	Access object key
`.key.nested`	Access nested key
`.[0]`	First array element
`.[]`	Iterate all array elements
`select(.age > 20)`	Filter by condition
`map(.name)`	Transform each element
`length`	Array length or string length
`keys`	Object keys as array

jq Examples

```
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)'
```

Common Patterns

API Response

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

Config File

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

Tips

`Validate`	Use jsonlint or python -m json.tool
`Pretty print`	jq . file.json or python -m json.tool
`JSONL`	One JSON object per line (newline-delimited)
`JSON5 / JSONC`	Extensions allowing comments and trailing commas