JSON Formatting Cheatsheet

A comprehensive guide to JSON formatting, covering syntax, data types, best practices, and tools for creating readable and maintainable JSON documents.

JSON Basics & Syntax

Core Concepts

JSON (JavaScript Object Notation): A lightweight data-interchange format that is easy for humans to read and write and easy for machines to parse and generate.

- Based on a subset of JavaScript syntax.
- Uses key-value pairs and ordered lists.
- Platform independent and widely supported.

Data Types: JSON supports several primitive data types:

- `string`: Unicode string, enclosed in double quotes.
- `number`: Integer or floating-point number.
- `boolean`: `true` or `false`.
- `null`: Represents an empty value.
- `object`: A collection of key-value pairs, enclosed in curly braces `{}`.
- `array`: An ordered list of values, enclosed in square brackets `[]`.

Syntax Rules

Key-Value Pairs

Keys must be strings enclosed in double quotes. Values can be any of the supported JSON data types.

Example:
`{"name": "John Doe", "age": 30}`

Objects

A collection of key-value pairs, enclosed in curly braces `{}`.

Example:
`{ "city": "New York", "country": "USA" }`

Arrays

An ordered list of values, enclosed in square brackets `[]`.

Example:
`[ "apple", "banana", "cherry" ]`

Nesting

JSON objects and arrays can be nested to represent complex data structures.

Example:

```
{
  "name": "Jane Doe",
  "address": {
    "street": "123 Main St",
    "city": "Anytown"
  }
}
```

Formatting Best Practices

Indentation

Use consistent indentation to improve readability. A common practice is to use 2 or 4 spaces for each level of indentation. Avoid using tabs as they can be interpreted differently by different editors.

Example (2 spaces):

```
{
  "name": "John",
  "age": 30
}
```

Example (4 spaces):

```
{
    "name": "John",
    "age": 30
}
```

Line Breaks

Insert line breaks after each comma to separate key-value pairs in objects and elements in arrays. This makes the structure easier to follow.

Example:

```
{
  "name": "John",
  "age": 30,
  "city": "New York"
}
```

Consistent Quotes

Always use double quotes for strings. JSON specification requires keys to be enclosed in double quotes as well.

Valid:
`{"name": "John"}`

Invalid:
`{'name': 'John'}` (single quotes are not valid)

Avoiding Trailing Commas

Do not include trailing commas after the last key-value pair in an object or the last element in an array. Trailing commas are invalid JSON and can cause parsing errors.

Invalid:

```
{
  "name": "John",
  "age": 30,
}
```

Valid:

```
{
  "name": "John",
  "age": 30
}
```

Advanced Formatting & Tools

JSON Validators

Use JSON validators to ensure your JSON documents are well-formed and valid. Validators can catch syntax errors, incorrect data types, and other issues.

Online Validators:

- JSONLint (jsonlint.com)
- JSONFormatter (jsonformatter.org)

Command-line Tools:

- `jq` (a lightweight and flexible command-line JSON processor)
- `python -m json.tool` (Python's built-in JSON validator)

JSON Formatters/Beautifiers

Use formatters to automatically indent and add line breaks to your JSON documents, making them more readable.

Online Formatters:

- JSONFormatter.org
- FreeFormatter.com

Text Editor Plugins:

- VS Code: Prettier, JSON Tools
- Sublime Text: Pretty JSON
- Atom: atom-beautify

Schema Validation

Use JSON Schema to define the structure and data types of your JSON documents. This helps ensure data consistency and can be used to validate JSON documents programmatically.

Key Concepts:

- `$schema`: Specifies the JSON Schema version.
- `type`: Defines the data type (e.g., `string`, `number`, `object`, `array`).
- `properties`: Defines the properties of an object and their types.
- `required`: Specifies which properties are mandatory.
- `enum`: Restricts a value to a predefined set of values.

Example:

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "type": "object",
  "properties": {
    "name": { "type": "string" },
    "age": { "type": "integer", "minimum": 0 }
  },
  "required": ["name", "age"]
}
```

Common Issues & Solutions

Encoding Issues

Ensure your JSON documents are encoded in UTF-8 to support a wide range of characters. Incorrect encoding can lead to parsing errors or data corruption.

Solution:

- Save your JSON files in UTF-8 encoding.
- Specify the encoding in the `Content-Type` header when sending JSON data over HTTP (`application/json; charset=utf-8`).

Escaping Special Characters

Special characters in strings, such as double quotes, backslashes, and control characters, must be escaped using backslashes.

Common Escape Sequences:

- `\"`: Double quote
- `\\`: Backslash
- `\/`: Forward slash
- `\b`: Backspace
- `\f`: Form feed
- `\n`: Newline
- `\r`: Carriage return
- `\t`: Tab
- `\uXXXX`: Unicode character (e.g., `\u00A9` for the copyright symbol)

Large Numbers

JavaScript's `Number` type can only accurately represent integers up to a certain limit (Number.MAX_SAFE_INTEGER). For larger numbers, consider using strings to avoid precision issues.

Example:

```
{
  "id": "12345678901234567890"  // Store large numbers as strings
}
```