Introduction to YAML

YAML Ain't Markup Language

What is YAML?

Human-readable data serialization format

- **Configuration files** (Docker, GitHub Actions)
- **Data exchange** between applications
- **Document headers** (Marp, Hugo, and Obsidian)

Basic Syntax Rules

- YAML has `key: value` property sets.
- Comments start with #

```yaml
name: "John Smith"           # String
age: 25                      # Number
is_student: true             # Boolean
graduation: null             # Null value
```

In JSON, an object is represented by `{}` with properties(key-value pairs).

```
{
  "name": "John Smith",
  "age": 25,
  "is_student": true,
  "graduation": null
}
```

Indentation and - makes an array

```
# Lists (arrays)
courses:
  - "Computer Science"
  - "Mathematics"
  - "Physics"

{
  "address": [
    Computer Science,
    "Mathematics",
    "Physics"
  ]
}
```

Indentations make an object

- In YAML, an object is a collection of properties.

```yaml
# Objects (key-value pairs)
address:
  street: "123 Main St"
  city: "Boston"
  zip: 02101
```

```json
{
  "address": {
    "street": "123 Main St",
    "city": "Boston",
    "zip": 02101
  }
}
```

Indentation with - makes an array, indentation without - makes an object.

```yaml
# An array with properties
address:
  - street: "123 Main St"
  - city: "Boston"
  - zip: 02101
```

```json
{
  "address": [
    {"street": "123 Main St"},
    {"city": "Boston"},
    {"zip": 02101}
  ]
}
```

Multi-line Strings

YAML - Multiple ways to handle text:

```yaml
description: |
  This is a multi-line
  description that preserves
  line breaks.

summary: >
  This is a long text
  that will be folded
  into a single line.
```

In Marp, to express a one continuous block of text (CSS), we use multi-line strings.

```
style: |
  strong {
    text-shadow: 2px 2px 4px #000000;
  }
```

If we use an array, it will be interpreted as three separate lines.

```
style:
  - strong {
  - text-shadow: 2px 2px 4px #000000;
  - }
```

JSON - Manual escape characters:

```
{
  "description": "This is a multi-line\ndescription that preserves\nline breaks."
}
```

Dart/Flutter uses YAML

```yaml
name: my_app
version: 1.0.0
dependencies:
  http: ^1.2.0
  json_annotation: ^4.9.0
assets:
  - assets/images/
  - assets/data/
```

If Dart uses JSON, it should be as follows:

```
{
  "name": "my_app",
  "version": "1.0.0",
  "dependencies": {
    "http": "^1.2.0",
    "json_annotation": "^4.9.0"
  },
  "assets": [
    "assets/images/",
    "assets/data/"
  ]
}
```

Node.js uses JSON

```
{
  "name": "my-node-app",
  "version": "1.0.0",
  "dependencies": {
    "express": "^4.19.0",
    "axios": "^1.6.7"
  },
  "scripts": [
    "start",
    "build",
    "test"
  ]
}
```

The equivalent YAML can be:

```yaml
name: my-node-app
version: 1.0.0
dependencies:
  express: ^4.19.0
  axios: ^1.6.7
scripts:
  - start
  - build
  - test
```

Differences from JSON

JSON arrays are ordered lists in [] containing values (string, number, object, array, boolean, or null), not standalone key-value pairs—causing confusion when converting to/from YAML.

```
{ WRONG JSON!!!
  "address": [
    "city": "Boston"
    "zip": 02101
  ]
}
```

```
{
  "address": [
    {"city": "Boston"},
    {"zip": 02101}
  ]
}
```

YAML allows lists of mappings, even if each mapping has only one key.

- In YAML, this is valid.
- It is an array with three properties, *not objects*.

```yaml
address:
  - city: "Boston"
  - zip: 02101
```

An array of two objects

```yaml
address:
  - street: "123 Main St"
    zip: 02101
  - street: "123 Second St"
    zip: 02102
```

```json
{
  "address": [
    {
      "street": "123 Main St",
      "zip": 2101
    },
    {
      "street": "123 Second St",
      "zip": 2102
    }
  ]
}
```

Real-World Usage Statistics

JSON Dominance:

- **90%+ of web APIs** use JSON
- **Native browser support**
- **Fastest parsing** in most languages
- **Smallest file size** for data exchange

YAML Growing Usage:

- **Docker Compose** - 100% YAML
- **Kubernetes** - YAML preferred
- **GitHub Actions** - YAML workflows
- **Ansible playbooks** - YAML only

Conclusion

Use Case	Recommended Format	Why
Web API	JSON	Speed, size, universal support
Config Files	YAML	Comments, readability
Data Storage	JSON	Performance, compatibility
Documentation	YAML	Human-friendly, comments
Mobile Apps	JSON	Minimal overhead
DevOps	YAML	Infrastructure as code standard

Feature	JSON	YAML
Syntax	Brackets & braces	Indentation-based
Readability	Good	Excellent
Comments	❌ Not allowed	✅ Supported
Data Types	6 basic types	Rich types + custom
File Size	Smaller	Larger
Parse Speed	Faster	Slower
Web APIs	✅ Standard	❌ Rare
Config Files	⚠️ Limited	✅ Preferred
Learning Curve	Easy	Moderate

Choose the Right Tool

JSON for data, YAML for configuration