YAML

YAML (YAML Ain’t Markup Language) is a human-readable data serialization language designed for simplicity and clarity. It is often used for configuration files, data exchange between programming languages, and declarative system descriptions (e.g., Docker Compose, GitHub Actions, Kubernetes).

History

YAML was first proposed in **2001** by **Clark Evans**, together with **Ingy döt Net** and **Oren Ben-Kiki**. The goal was to create a format that combined the **readability of plain text** with the **structure of JSON or XML**, making it easy for humans to write and understand while remaining machine-parsable. The acronym originally meant *“Yet Another Markup Language”*, but was later reinterpreted humorously as *“YAML Ain’t Markup Language”*, to emphasize that YAML focuses on **data**, not **documents** or **markup**.

Basic Idea

YAML is based on **indentation and key-value pairs**, allowing hierarchical (tree-like) data structures without the need for braces or brackets. It is often described as a **human-friendly alternative** to JSON and XML.

Comparison with JSON

Concept	JSON	YAML
Syntax	Uses braces `{}` and brackets `[]`	Uses indentation (spaces only)
Comments	Not allowed	Allowed with `#`
Readability	Machine-friendly	Human-friendly
Common use	APIs, web data exchange	Configuration, DevOps, CI/CD

Example comparison:

```
{
  "student": {
    "name": "Anna",
    "age": 21,
    "courses": ["Programming", "Databases"]
  }
}
```

```
student:
  name: Anna
  age: 21
  courses:
    - Programming
```

```
    - Databases
```

Syntax Rules

- Indentation defines structure (use spaces, not tabs)
- Key-value pairs: `key: value`
- Lists: prefix `-`
- Nested structures: indent by two spaces
- Comments: start with `#`

Example:

```yaml
server:
  host: localhost
  port: 8080
  enabled: true
  paths:
    - /login
    - /logout
```

Data Types

YAML supports a range of basic and complex data types. Values can be written in **implicit** or **explicit** form — YAML automatically detects the type from context, but types can also be specified manually using tags (e.g., `!!str`, `!!int`).

1. Scalars

Scalars are single values such as strings, numbers, or booleans.

Type	Example	Notes
String	`name: “Alice”`	Quotation marks are optional unless special characters are used.
Integer	`age: 25`	No quotes needed; negative values allowed.
Float	`price: 19.99`	Decimal notation or scientific form (`1.2e+3`) supported.
Boolean	`enabled: true` or `enabled: no`	`true/false`, `yes/no`, and `on/off` are equivalent.
Null	`value: null` or `value: ~`	Both mean “no value”.
Date/Time	`created: 2025-11-03`	ISO 8601 format is recommended.

Explicit typing (less common but useful for validation):

```yaml
id: !!int "42"
flag: !!bool "yes"
pi: !!float "3.14159"
text: !!str 1234   # forced as string, not number
```

2. Strings

YAML offers flexible ways to define strings:

- **Plain style:** `title: Hello World`
- **Single-quoted:** `path: 'C:\Users\Name'`

(backslashes are preserved literally)

- **Double-quoted:** `message: “Line1\nLine2”`

(supports escape sequences like `\n`, `\t`)

- **Multi-line literal (`|`):** preserves line breaks

```
description: |
  This is line one.
  This is line two.
```

- **Folded block (`>`):** joins lines into a single paragraph

```
note: >
  This sentence
  continues on the next line.
```

3. Collections

YAML supports two structured types: **sequences (lists)** and **mappings (dictionaries)**.

- **Sequences:** ordered lists of elements, marked with `-`

```
colors:
  - red
  - green
  - blue
```

- **Mappings:** unordered key–value pairs

```
person:
  name: Bob
  age: 30
  city: London
```

- **Inline form:** lists and dictionaries can also be written on one line

```
colors: [red, green, blue]
```

```
    person: {name: Bob, age: 30}
```

4. Nested Structures

Lists and mappings can be combined to represent complex hierarchical data:

```
students:
  - name: Anna
    grades: [A, B, A]
  - name: Mark
    grades:
      - B
      - C
      - A
```

5. Aliases and Anchors

YAML allows referencing the same data in multiple places using **anchors (&)** and **aliases (*)**.

```
defaults: &base
  host: localhost
  port: 8080

development:
  <<: *base
  debug: true
```

This feature reduces duplication and keeps configuration files consistent.

6. Summary

- YAML automatically infers most types but supports explicit typing.
- Scalars, sequences, and mappings cover all standard data models.
- Multi-line and folded strings improve readability.
- Anchors and aliases allow reuse of data blocks.

Validation and Schema

Just like JSON Schema, YAML files can be validated using schema definitions. Common tools include **Yamale**, **Kubeval**, or the built-in schema support of IDEs such as Visual Studio Code.

Typical Use Cases

- **Docker Compose** (`docker-compose.yml`)
- **GitHub Actions** (`.github/workflows/*.yml`)
- **Kubernetes manifests** (`deployment.yaml`)
- **Python and Node.js configuration files**

Example:

```yaml
version: "3.8"
services:
  web:
    image: nginx:latest
    ports:
      - "8080:80"
```

Educational Demo Idea

Show the same configuration both in JSON and YAML, and ask:

- Which one is easier to read?
- What are the risks of using indentation as syntax?
- How does the structure represent a **syntax tree**?

Summary

- YAML is a **readable, indentation-based** language for structured data.
- It was created to bridge the gap between human readability and machine processing.
- It plays a central role in modern **DevOps**, **configuration management**, and **data description languages**.