eazy bytes

YAML

ZERO TO MASTER

YAML ZERO TO MASTER

eazy bytes

WHAT WE COVER IN THIS COURSE

Introduction to YAML

- What is YAML
- YAML Use cases
- Writing Sample YAML file
- XML vs JSON vs YAML

YAML Basic concepts

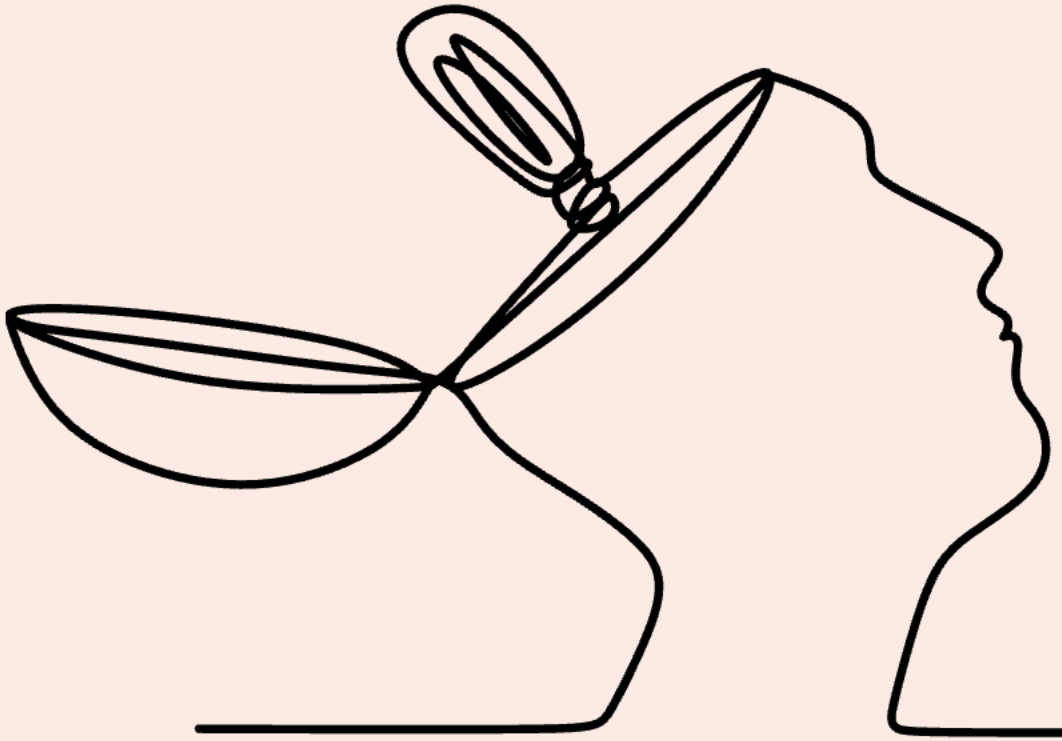
- Scalars, Strings, Sequences, Dictionaries
- Comments
- Dates & Timestamps
- Tags

YAML Advance concepts

- Sequence/Collections
- Sets, Ordered Mapping
- Writing Complex keys
- Anchors, Alias, Overriding
- Multi documents support

eazy bytes

What is YAML ?



YAML (YAML Ain't Markup Language) is a human-readable data serialization standard that can be used in conjunction with all programming languages and is often used to write configuration files.

It is designed to be easy to read and write, making it a popular choice for configuration management.

What is YAML ?

eazy bytes

Human-Readable & Hierarchical

YAML is designed to be easy for humans to read and write, using indentation to represent structure and a minimalistic syntax. YAML structures data hierarchically, using indentation to denote levels of depth.

Flexible Data Types

YAML supports various data types including scalars (strings, integers, floats, booleans), lists, sets and dictionaries (mappings).

Cross-Platform

YAML can be used across different programming languages, making it a versatile choice for configuration files.

Common Uses

YAML is often used for configuration files, data exchange between languages with different data structures, and to write automation scripts.

eazy bytes

YAML Usecases

YAML is widely used in various applications, including:

Configuration Management: Tools like Ansible, Kubernetes, and Docker Compose use YAML for their configuration files.

Data Storage: Some applications store small amounts of data in YAML files for ease of access and readability.

Data Serialization: YAML can be used to serialize data for communication between systems.

Overall, YAML's simplicity and readability make it a popular choice for many use cases, especially in the realm of configuration and automation.

eazy bytes

YAML Usecases

Since YAML is a human friendly readable format, it is widely used in writing configurations files in different DevOps tools, cloud platforms and applications.

BELOW ARE THE FEW OF THE MOST FAMOUS TOOLS THAT USES YAML HEAVILY

Docker

Kubernetes

Prometheus

Grafana Labs

Jenkins

ANSIBLE

CHEF Progress

puppet

Terraform

github

spring

circleci

Travis CI

Red Hat

aws

Microsoft Azure

Google Cloud Platform

Sample YAML file

eazy bytes

sample.yml file representing person details

```yaml
person:
  name: Madan Reddy
  age: 35
  gender: M
  is_student: false
  hobbies:
    - reading
    - travelling
    - hiking
  address:
    street: 123 Main St
    city: SomeCity
    postal_code: 12345
```

In this example:

- person is a dictionary with keys name, age, gender, is_student, hobbies, and address
- name is a string
- age is an integer
- is_student is a boolean
- hobbies is a list of strings
- address is a nested dictionary

Sample YAML file

eazy bytes

YAML extensions

YAML files are created with extensions

".yaml " or yml "

You can use any IDE or text editor to open/create YAML files.

Indentation

Indentation of whitespace is used to denote structure. This is very similar to Python uses indentation to highlight the blocks of code.

YAML is **case sensitive** in nature.

[name: madan != Name: Madan]

Map structure

The basic structure of a YAML file is a map. You might call this a dictionary, hash or object, depending on your favorite programming language.

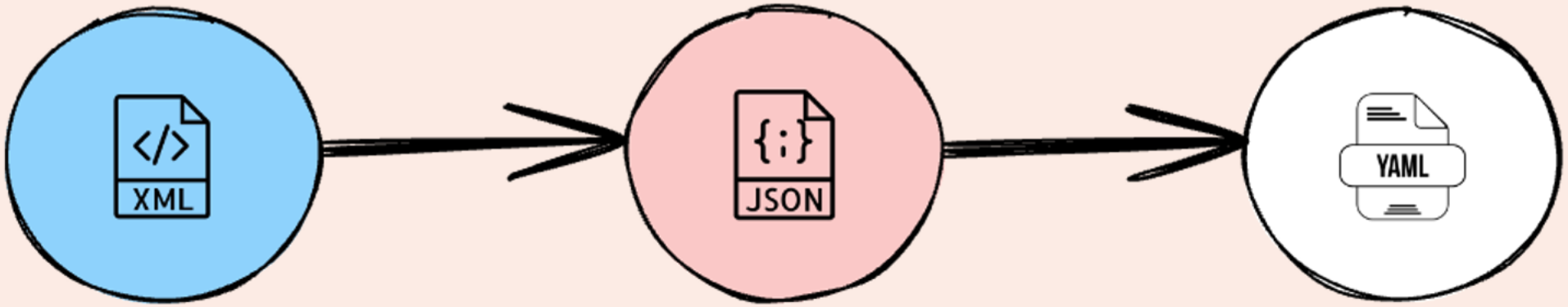
YAML is not a programming languagge

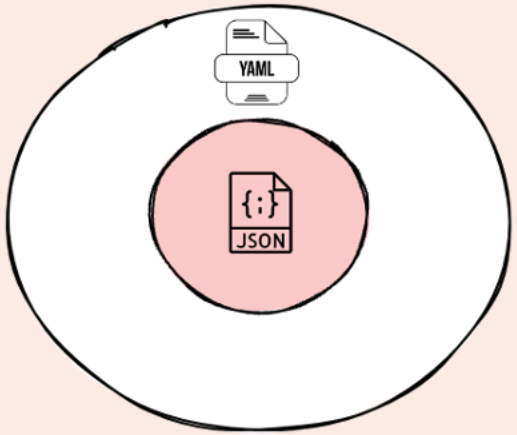
YAML is a superset of JSON, meaning valid JSON is also valid YAML

XML vs JSON vs YAML

eazy bytes

Evolution of XML, JSON & YAML





YAML is a superset of JSON, meaning it includes all the features of JSON and more. To draw an analogy with programming languages, YAML is to JSON what TypeScript is to JavaScript. Just as TypeScript builds upon JavaScript by adding additional features, YAML extends JSON with more capabilities while maintaining full compatibility.

XML vs JSON vs YAML

eazy bytes

XML - eXtensible Markup Language

```xml
<person>
  <name>Madan Reddy</name>
  <age>35</age>
  <gender>M</gender>
  <is_student>false</is_student>
  <hobbies>
    <hobby>reading</hobby>
    <hobby>travelling</hobby>
    <hobby>hiking</hobby>
  </hobbies>
  <address>
    <street>123 Main St</street>
    <city>SomeCity</city>
    <postal_code>12345</postal_code>
  </address>
</person>
```

XML is like your very formal, old-fashioned friend who insists on using the proper etiquette at all times. They speak in full sentences with lots of flourish and formality.

JSON - JavaScript Object Notation

```json
{
    "person": {
        "name": "Madan Reddy",
        "age": 35,
        "gender": "M",
        "is_student": false,
        "hobbies": [
            "reading",
            "travelling",
            "hiking"
        ],
        "address": {
            "street": "123 Main St",
            "city": "SomeCity",
            "postal_code": 12345
        }
    }
}
```

JSON is your cool, laid-back friend who likes to keep things simple and straightforward. They're modern, efficient, and don't like to waste time on unnecessary details.

YAML - YAML Aint Markup Language

```yaml
person:
  name: Madan Reddy
  age: 35
  gender: M
  is_student: false
  hobbies:
    - reading
    - travelling
    - hiking
  address:
    street: 123 Main St
    city: SomeCity
    postal_code: 12345
```

YAML is your easygoing, minimalist friend who believes in the beauty of simplicity and whitespace. They get straight to the point without any fuss.

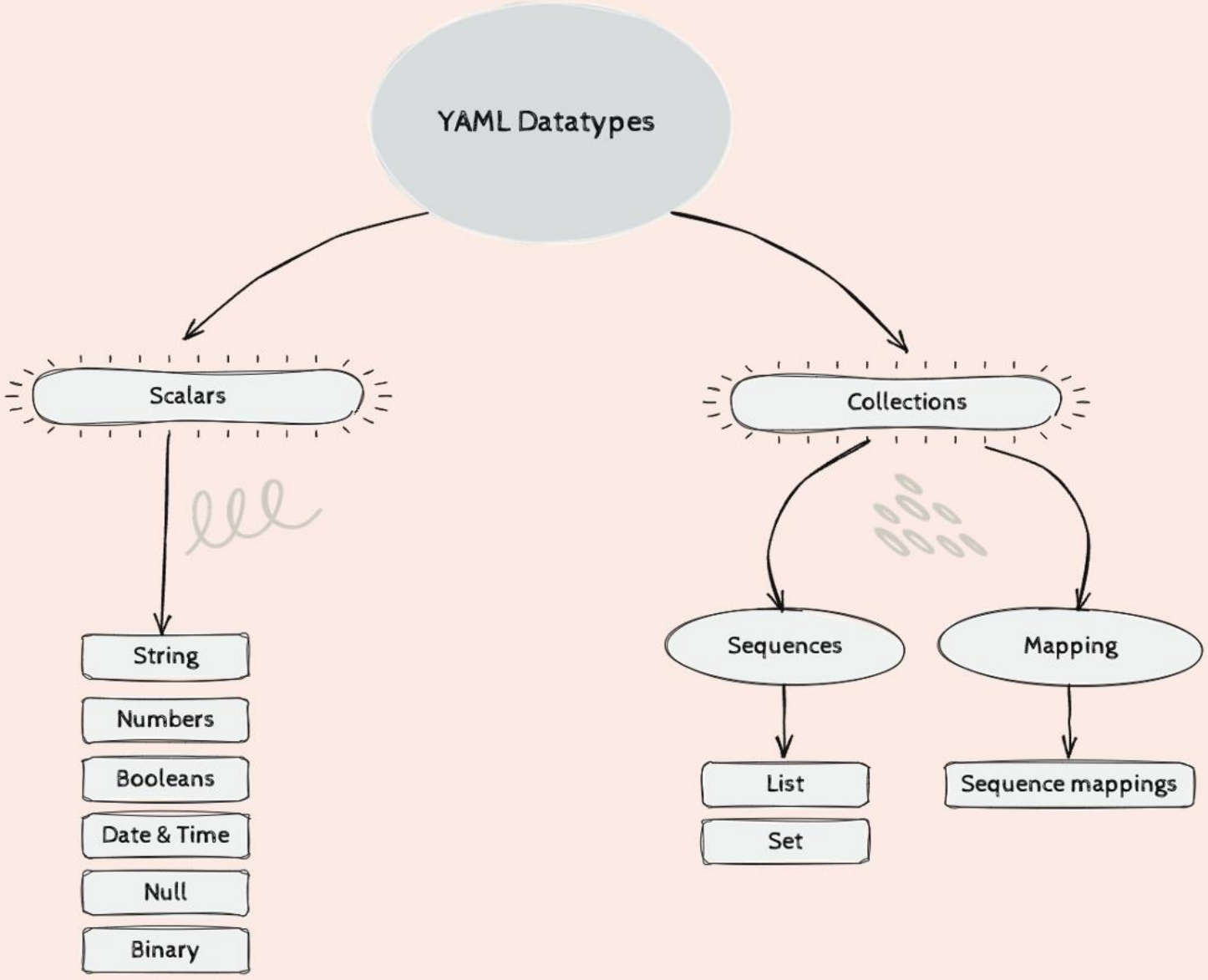
XML vs JSON vs YAML

eazy bytes

	XML	JSON	YAML
HUMAN-READABLE	Harder to read	Harder to read	Easier to read and understand
SYNTAX	More verbose	Explicit, strict syntax requirements	Minimalist syntax
COMMENTS	Allows comments	Comments are not allowed	Allows comments
HIERARCHY	Hierarchy is denoted by using open & close tags.	Hierarchy is denoted by using braces and brackets.	Hierarchy is denoted by using space or tab characters.
STORAGE	Will take lot of storage and network bandwidth.	Lighter compared to XML	Lighter compared to XML & JSON
USECASES	Best for complex projects that require fine control over schema	Favored in web development & transmitting data over HTTP	Best for configuration files, along with JSON features.

YAML Datatypes

eazy bytes



Scalars in YAML

eazy bytes

Scalars in YAML represent single, indivisible values. They are the most basic data type in YAML and can include strings, numbers, booleans, null values, and other simple values. Scalars do not contain any nested structures, such as lists or dictionaries, which are more complex data types in YAML.

Scalars represent a single stored value. Scalars are assigned to key names as values.
You define a key with a name, colon, and space, then a value for it to hold

<key>: <value>

Example

name: Madan

Represents Scalar

Space is mandatory between a Key and Value after colon. With out space it will be treated as a single String key value

Scalars in YAML

eazy bytes

```yaml
fname: "Madan"
lname: Reddy
gender: 'M'
age: 35
weight: 68.5
is_student: false
isAsian: Yes
hexNum: 0x12d6
octalNum: 0o24432
positive-infinity: .inf
negative-infinity: -.Inf
invalidNumber: .NAN
null-value: null
null-value1: ~
null-value2:
```

Strings

integer number

float number

boolean

String from 1.2.x, boolean in <1.2.x

int

int

float

float

NoneType

Strings in YAML

eazy bytes

In YAML, strings can be written in several ways: without quotes (plain style), with single quotes, or with double quotes. Each method has its own use cases and considerations. Here's a breakdown of when to use each approach:

Plain Style (No Quotes) - Use plain style for simple strings without special characters, spaces or keywords that could be misinterpreted by the YAML parser.

```
name: JohnDoe
age: 30
```

Single-quoted Style - Use single quotes when you need to include special characters, spaces or reserved keywords, but do not need to process escape sequences.

```
message: 'Hello, World!'
isAdmin: 'true'
special: 'text with : colon'
```

Single quotes allow you to include any character except a single quote itself. To include a single quote within the string, use two single quotes (‘’).

Single quotes do not interpret escape sequences (e.g., \n remains as is).

Strings in YAML

eazy bytes

Double-quoted Style - Use double quotes when you need to include special characters, spaces, reserved keywords, or need to process escape sequences.

```
greeting: "Hello, World!"
special_characters: "text with : colon and # hash"
multiline: "Line 1\nLine 2\nLine 3”
```

Double quotes allow for the inclusion of escape sequences like ‘\n’ (newline), ‘\t’ (tab), ‘\”’ (double quote), etc.

Defining Long/multi-line Strings in YAML

eazy bytes

In YAML, the | and > symbols, along with the optional + and - modifiers, are used to define long strings. These symbols help to manage how newlines and trailing spaces are treated.

Literal Block Style (|) - The '|' symbol is used to preserve newlines exactly as they are written in the YAML file. This is useful when you want the text to be preserved verbatim.

```
literal_block: |
  This is a literal block.
  Newlines are preserved.
  So is indentation.
  Trailing spaces are preserved as well.
```

output

```
This is a literal block.
Newlines are preserved.
So is indentation.
Trailing spaces are preserved as well.
```

Defining Long/multi-line Strings in YAML

eazy bytes

Folded Block Style (>) - The '>' symbol is used to fold newlines into spaces. This is useful for text where you don't want hard line breaks to appear in the output.

```
folded_block: >
  This is a folded block.
  Newlines become spaces.
  Indentation is not preserved.
  Trailing spaces are removed.
```

output

```
This is a folded block. Newlines become spaces. Indentation is not preserved. Trailing spaces are removed.
```

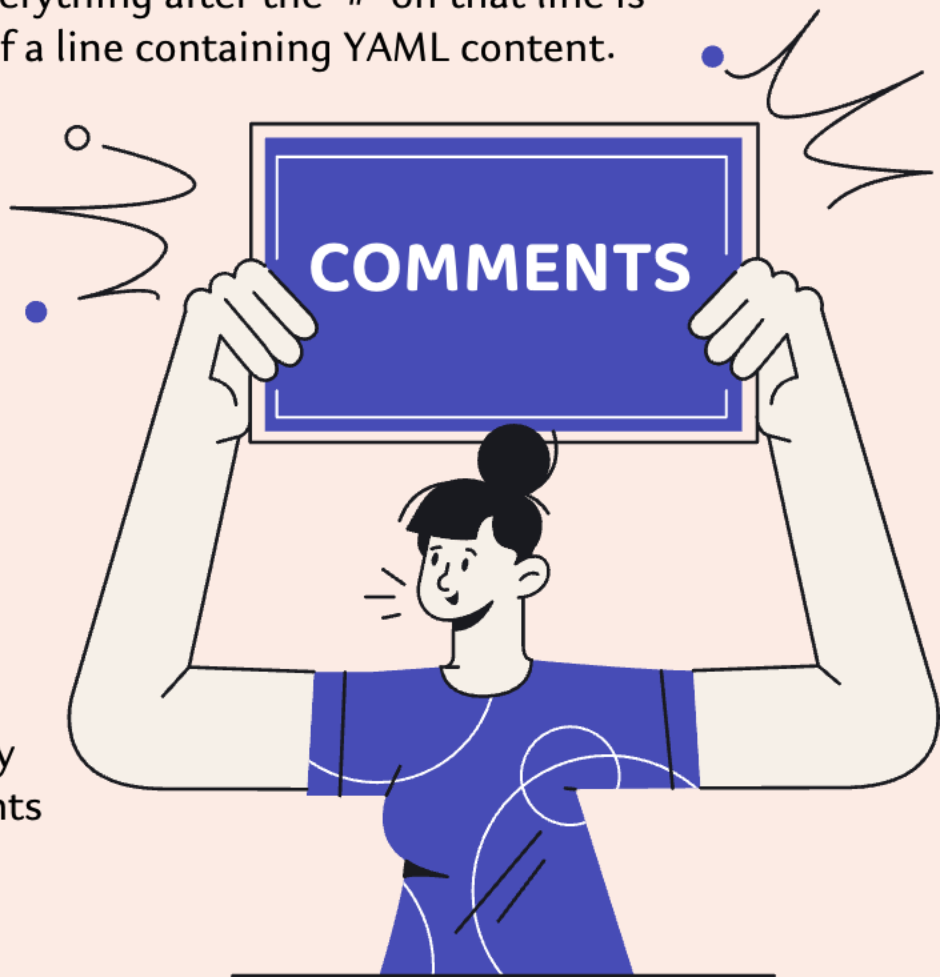
Comments in YAML

eazy bytes

YAML **comments** are used to add explanatory notes or annotations to the YAML file. These comments are ignored by the YAML parser and are meant to provide context or explanations for humans who read the file.

Single-Line Comments: Comments in YAML are denoted by the '#' symbol. Everything after the '#' on that line is considered a comment. Comments can be placed on their own line or at the end of a line containing YAML content.

```
# This is a comment
key: value  # This is an inline comment
```



Multiline Comments: YAML does not support multiline comments directly. However, you can achieve a similar effect by using multiple single-line comments.

```
# This is a multiline comment
# It spans multiple lines
# Each line starts with a #
key: value
```

YAML and JSON Compatibility: When using YAML for configurations that may be converted to JSON, remember that JSON does not support comments. Comments will be lost in the conversion.

Comments in YAML

eazy bytes

Document Headers: Comments can be used at the beginning of a YAML file to provide a document header or metadata.

```yaml
# Configuration for the MyApp application
# Version 1.0.0
# Last updated: 2024-07-22
application:
  name: MyApp
  version: 1.0.0
```

Section Headers: Use comments to divide your YAML file into logical sections, especially for larger files.

```yaml
# Database configuration
database:
  host: localhost
  port: 5432
  username: user
  password: pass

# API configuration
api:
  endpoint: https://api.example.com
  key: api_key
```

Comments in YAML

eazy bytes

Commenting Out Code: Temporarily disable a part of the configuration by commenting it out.

```yaml
# This is the old configuration
# database:
#   host: oldhost
#   port: 1234

# New configuration
database:
  host: newhost
  port: 5432
```

Commenting in Nested Structures: Ensure comments in nested structures are properly indented to maintain readability.

```yaml
database:
  # Connection settings
  host: localhost
  port: 5432
  # Credentials
  username: user
  password: pass
```

Comments in YAML

eazy bytes

CI/CD Pipeline Configuration Example:

```yaml
# CI/CD pipeline configuration for building and deploying the application

# Stages of the pipeline
stages:
  - build  # Build the application
  - test   # Run tests
  - deploy # Deploy the application

# Build stage
build:
  script:
    - echo "Building the application..."
    - ./gradlew build

# Test stage
test:
  script:
    - echo "Running tests..."
    - ./gradlew test

# Deploy stage
deploy:
  script:
    - echo "Deploying the application..."
    - ./deploy.sh
```

Dates & Timestamps in YAML

eazy bytes

Dates and timestamps in YAML are used to represent date and time values. YAML recognizes certain patterns and automatically interprets them as dates or timestamps.

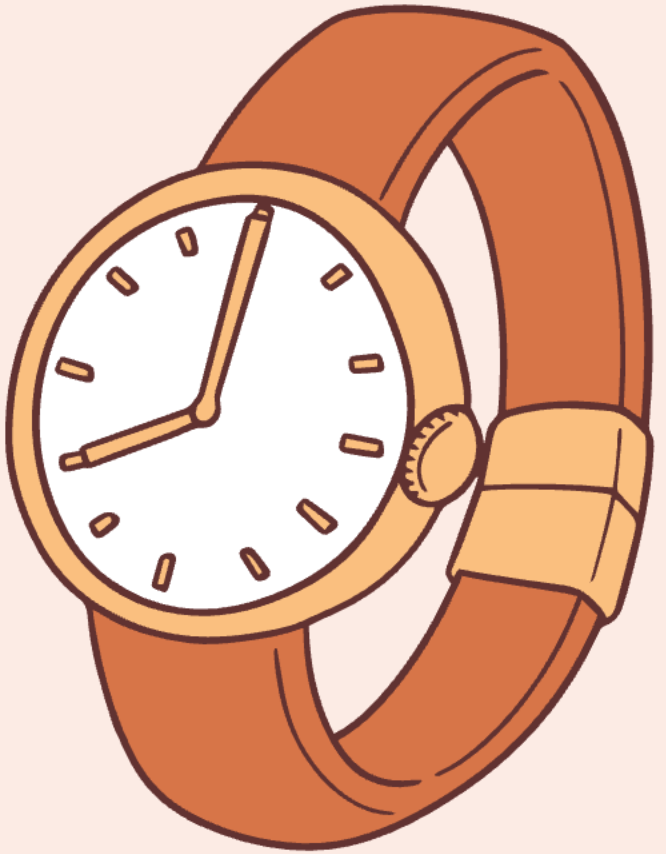
Standard Date Format: YYYY-MM-DD

```
someDate: 2034-07-23
```

Full Date & Time Format (Canonical) : YYYY-MM-DDTHH:MM:SSZ

T separates the date and time.
Z indicates UTC time.

```
timestamp: 2034-07-23T10:20:30Z
```



Dates & Timestamps in YAML

eazy bytes

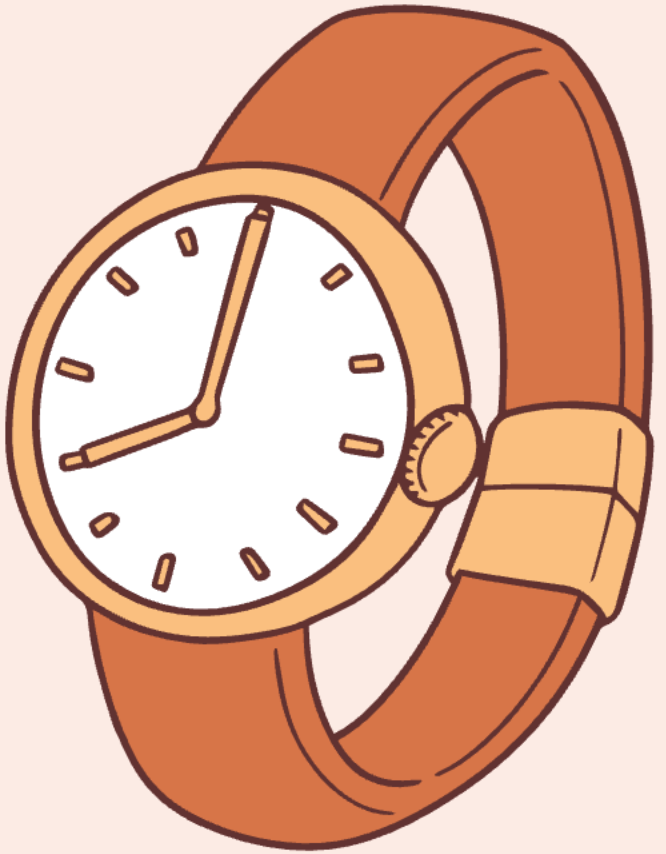
Full Date and Time with Timezone (iso8601): YYYY-MM-DDTHH:MM:SS±HH:MM

The timezone offset can be positive or negative.

```
timestamp_with_timezone: 2034-07-23T10:20:30+02:00
```

It is also possible to define full date and time with the following formats,

```
spacedtimestamp_with_timezone: 2034-07-23 10:20:30 -5
noTimeZone: 2034-07-23 10:20:30
```

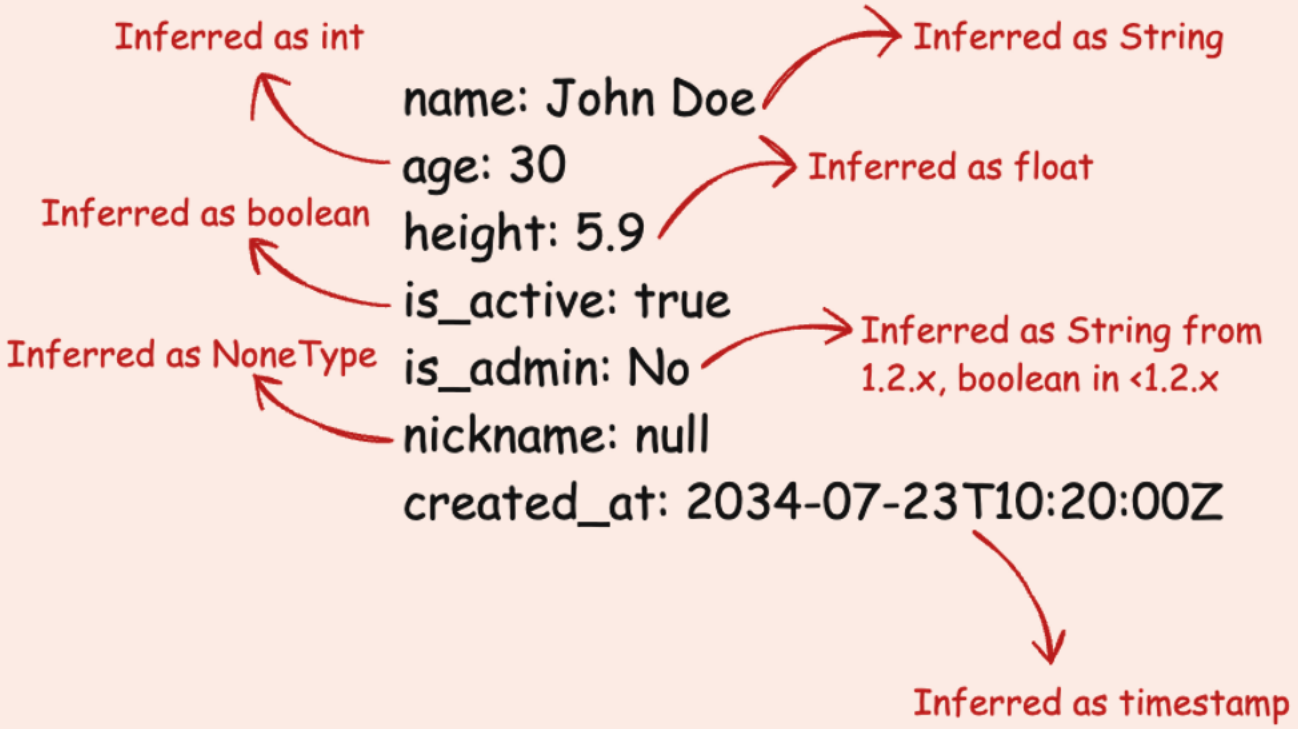


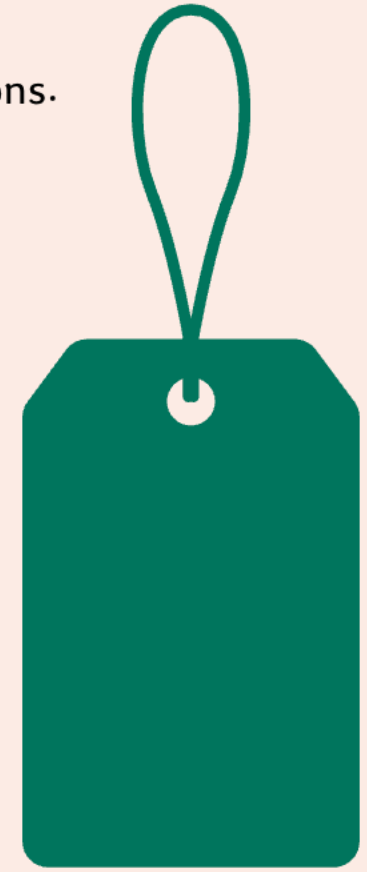
Tags in YAML

eazy bytes

In YAML, types can be defined both implicitly and explicitly. Understanding implicit and explicit typing, as well as tags, helps to control how YAML parsers interpret the data.

Implicit Typing means that the type of a value is inferred from its format without any additional annotations. YAML parsers recognize certain patterns and automatically assign types to values.

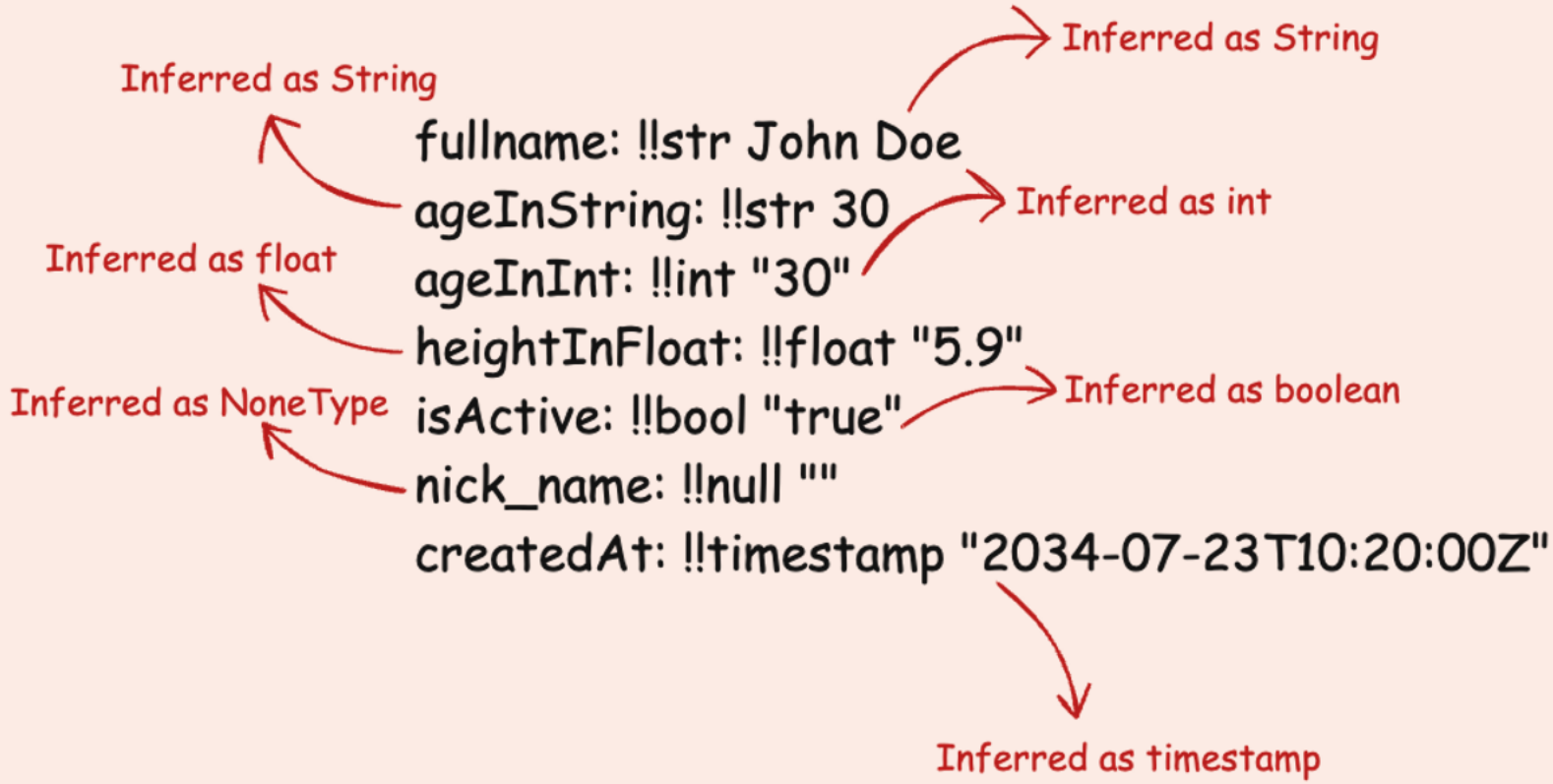


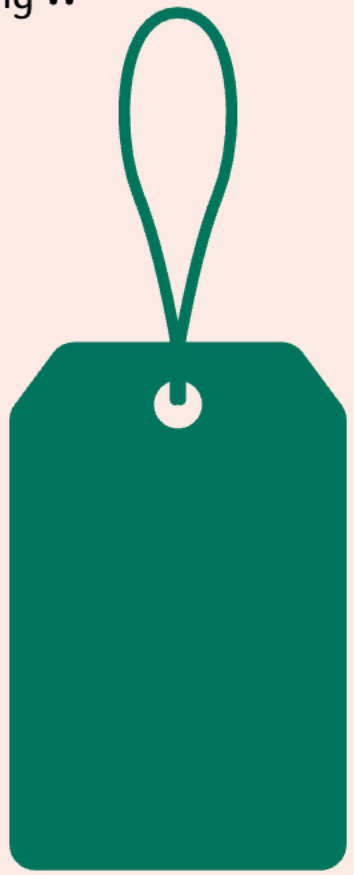


Tags in YAML

eazy bytes

Explicit typing involves using **tags** to specify the type of a value explicitly. This removes ambiguity and ensures that the value is interpreted in the intended way. Tags in YAML are used to specify the type of the value explicitly using **!!**





Sequences/Collections in YAML

eazy bytes

In YAML, sequences (also known as collections or lists) are used to represent ordered lists of items. YAML provides two primary styles for defining sequences: block style and flow style.

Block style is the default and more human-readable way of writing sequences in YAML. It uses indentation to define the hierarchy of elements. Block sequences indicate each entry with a dash and space ("- ").

```
hobbies:
  - reading
  - travelling
  - hiking
```

Flow style is more compact and JSON-like. It uses square brackets [] to define sequences.

```
hobbies: [reading, travelling, hiking]
```

Both block and flow styles support nested sequences and if needed, we can mix both block and flow styles

```
students:
 - name: John Doe
   age: 16
   favorites: {subjects: [Math, Science, History]}

 - name: Jane Smith
   age: 17
   favorites: {subjects: [Literature, Art, Biology]}
```

Sequences mappings in YAML

eazy bytes

In YAML, sequence mappings combine sequences (lists) and mappings (dictionaries) to create complex nested structures.

Sequence of Mappings

A sequence of mappings is a list where each item is a dictionary. This is useful for representing collections of objects with similar properties, such as a list of users or configuration settings.

Block Style:

```
users:
  - name: John Doe
    age: 30
    email: john.doe@example.com

  - name: Jane Smith
    age: 25
    email: jane.smith@example.com
```

Flow Style:

```
users: [
  {name: John Doe, age: 30, email: john.doe@example.com},
  {name: Jane Smith, age: 25, email: jane.smith@example.com}
]
```

Sequences mappings in YAML

eazy bytes

Mapping of Sequences

A mapping of sequences is a dictionary where each key maps to a list of values. This structure is useful for grouping related lists under different categories.

Block Style:

```
departments:
 engineering:
   - Alice
   - Bob
   - Carol

 marketing:
   - Dave
   - Eve
   - Frank
```

You can combine sequences of mappings and mappings of sequences to create complex nested structures.

Flow Style:

```
departments: {engineering: [Alice, Bob, Carol], marketing: [Dave, Eve, Frank]}
```

Sets in YAML

eazy bytes

In YAML, a set is a unique collection of values without duplicates. Sets are represented as a mapping where each key is associated with a null value.

Sets in YAML: Use the !!set tag.
Syntax: Use ? for each item in the set without corresponding values.
Usage: Ideal for collections where uniqueness is essential.

Ordered Mappings in YAML

eazy bytes

In YAML, ordered mappings can be represented using the **!!omap** tag, which ensures that the order of key-value pairs is preserved as specified. This can be particularly useful in scenarios where the order of entries matters, such as configuration files where the sequence of directives needs to be maintained.

Ordered Mappings in YAML: Use the !!omap tag.
Syntax: Each key-value pair is represented as a single-item mapping within a sequence.
Usage: Ensures the order of entries is preserved, useful for configuration files and ordered directives.

!!omap: This tag explicitly specifies that the data structure is an ordered mapping.

```
capitals: !!omap
  - USA: Washington DC
  - United Kingdom: London
  - India: Delhi
```

Each entry in the ordered mapping is represented as a single-item mapping within a sequence.

Common Structure of YAML

eazy bytes

```yaml
company:
  name: Example Corp
employees:
  - name: Alice Smith
    title: Software Engineer
    skills:
      - Python
      - JavaScript
      - YAML
  - name: Bob Johnson
    title: DevOps Engineer
    skills:
      - Docker
      - Kubernetes
      - Terraform
departments:
  - IT
  - HR
  - Sales
location: {city: Techville, state: TX} # Flow style
```

Scalars represent a single stored value Scalars are assigned to key names

Dictionaries/Mappings are collections of key value pairs all nested under the same subgroup

Sequences are data structures similar to a list or array that hold multiple values under the same key

Complex keys in YAML

eazy bytes

In YAML, keys in mappings can be complex data structures, not just simple strings. Complex keys can include long string, sequences (lists) and mappings (dictionaries).

We can use ? followed by a space to indicate the start of a complex key. Below are few examples,

A long complex string key is represented using ?

A long complex string key along with the new lines is represented using ? |

A sequence as a key is mentioned using ?

```
? Which is your favourite sport
  when you are a child ?
: Cricket
? |
  Which is your favourite sport
  when you are a child ?
: Cricket
? - NewYork
  - Chicago
  - Washington
: USA
? - Development
  - UAT
  - Production
: - https://dev.example.com
  - https://uat.example.com
  - https://example.com
```

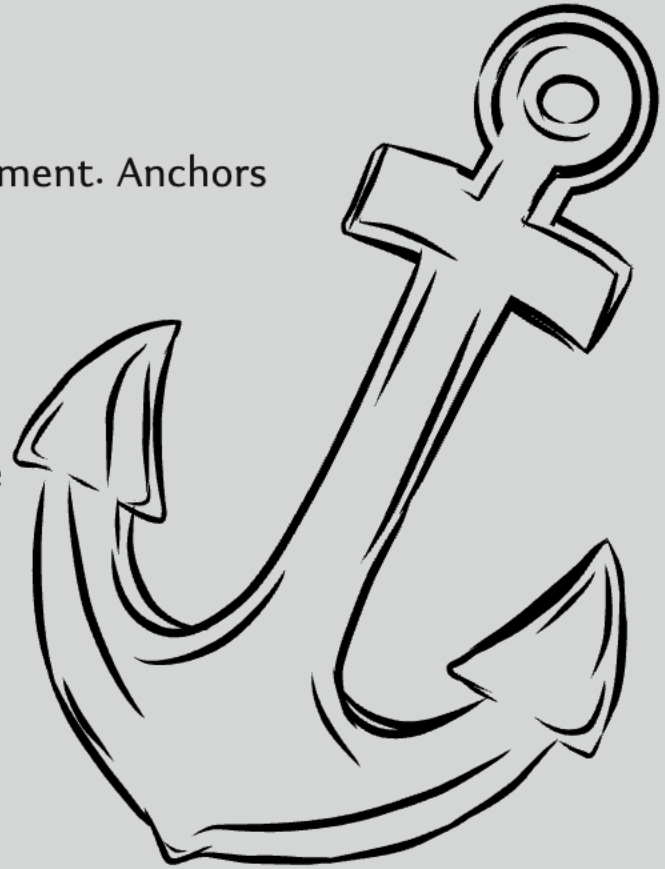
Anchors & Alias in YAML

eazy bytes

Anchors and aliases in YAML are powerful features that help manage repeated data and maintain consistency across your YAML documents. They allow you to define a piece of data once and reference it multiple times, making your YAML files more concise and easier to maintain.

Anchors

An anchor is a mechanism for defining a piece of data that can be reused later in the document. Anchors are defined using the & symbol followed by a name.



Aliases

An alias is a reference to an anchor. Aliases are defined using the * symbol followed by the anchor name. They allow you to reuse the anchored data elsewhere in the YAML document.

The Alias essentially acts as a "see above'' command, which makes the program pause standard traversal, return to the anchor point, then resume standard traversal after the Anchored portion is finished

Anchors & Alias in YAML

eazy bytes

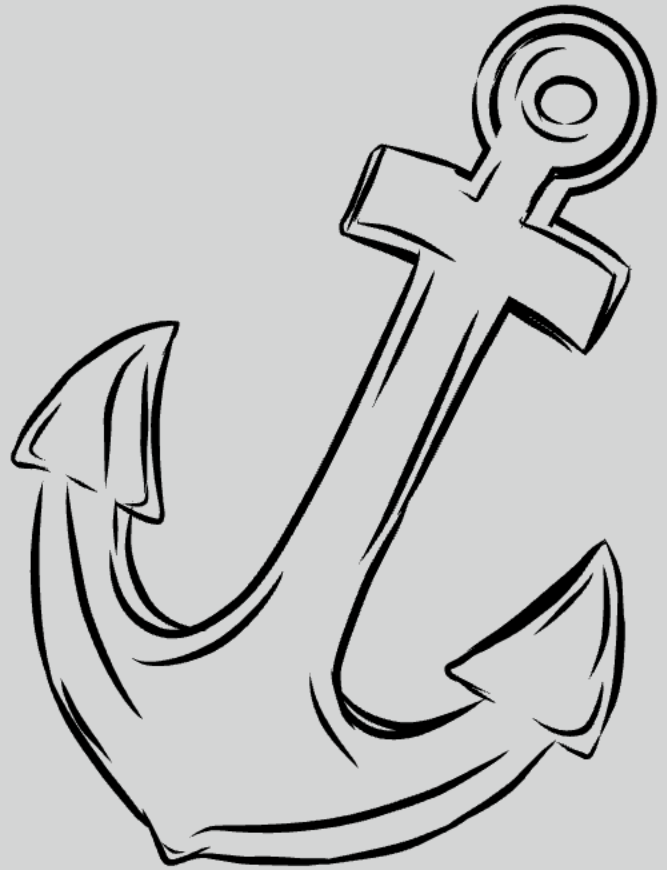
```
definitions:
  days:
    - weekday: &working
        wakeup: 6:00 AM
        activites:
          - workout
          - work
          - sleep
        sleeptime: 10:00 PM
    - weekend: &holiday
        wakeup: 8:00 AM
        activites:
          - workout
          - movies
          - sleep
        sleeptime: 12:00 AM

schedules:
  days:
    weekdays:
      - monday: *working
      - tuesday: *working
      - wednesday: *working
      - thursday: *working
      - friday: *working
    weekends:
      - saturday: *holiday
      - sunday: *holiday
```

An anchor with the name 'working' is created for the entire content under weekday

An anchor with the name 'holiday' is created for the entire content under weekend

The anchor 'working' is being aliased under the weekday elements

The anchor 'holiday' is being aliased under the weekend elements



Override Anchor values in YAML

eazy bytes

After defining anchor values inside your YAML file, what if you want essentially the same block of code with one small change? You can use **override** with the characters **'<<:'** before the Alias to add more values, or override existing ones.

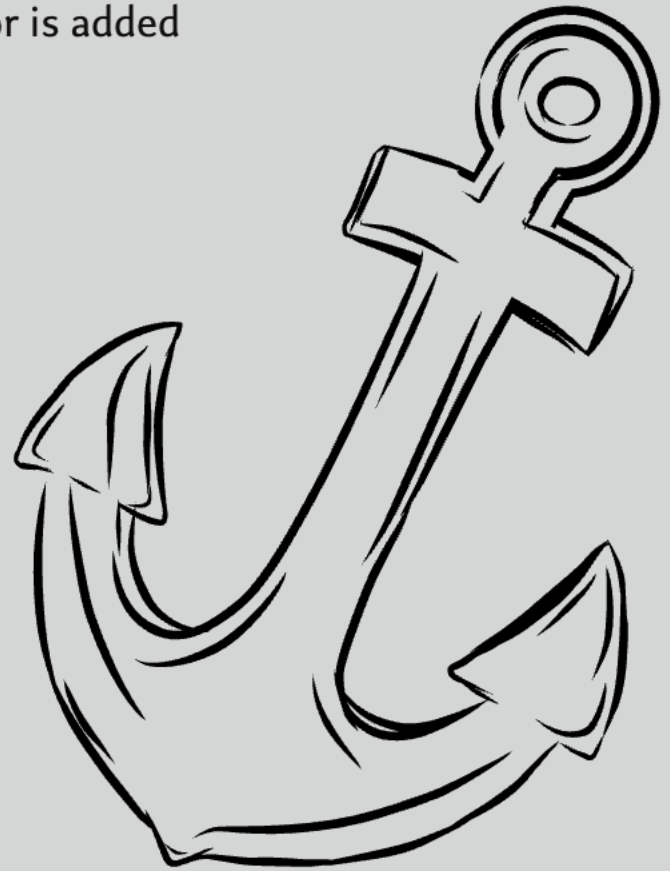
When using overrides, mappings are overridden if the new mapping has the same name or is added afterward if different. Overriding is also called as **merge**

```
schedules:
  days:
    weekdays:
      - monday: *working
      - tuesday: *working
      - wednesday: *working
      - thursday: *working
      - friday:
          <<: *working
          sleeptime: 12:00 AM
          netflix: true
    weekends:
      - saturday:
          <<: *holiday
          netflix: true
      - sunday:
          <<: *holiday
          sleeptime: 10:00 PM
```

overriding friday data with new sleeptime value and new netflix element

overriding saturday data with new netflix element

overriding sunday data with new sleeptime value



Multi Document support in YAML

eazy bytes

YAML supports the inclusion of multiple documents within a single YAML file. Each document in a YAML file is separated by a line containing three hyphens (---). Optionally, three dots (...) can be used to indicate the end of a document, though it's not strictly necessary.

Basic Structure

```
# First Document
---
key1: value1
key2: value2

# Second Document
---
keyA: valueA
keyB: valueB

# Third Document
---
keyX: valueX
keyY: valueY
```

Optional Document End Indicator

```
# First Document
---
key1: value1
key2: value2
...

# Second Document
---
keyA: valueA
keyB: valueB
...
```

eazy bytes

THANKYOU & CONGRATULATIONS