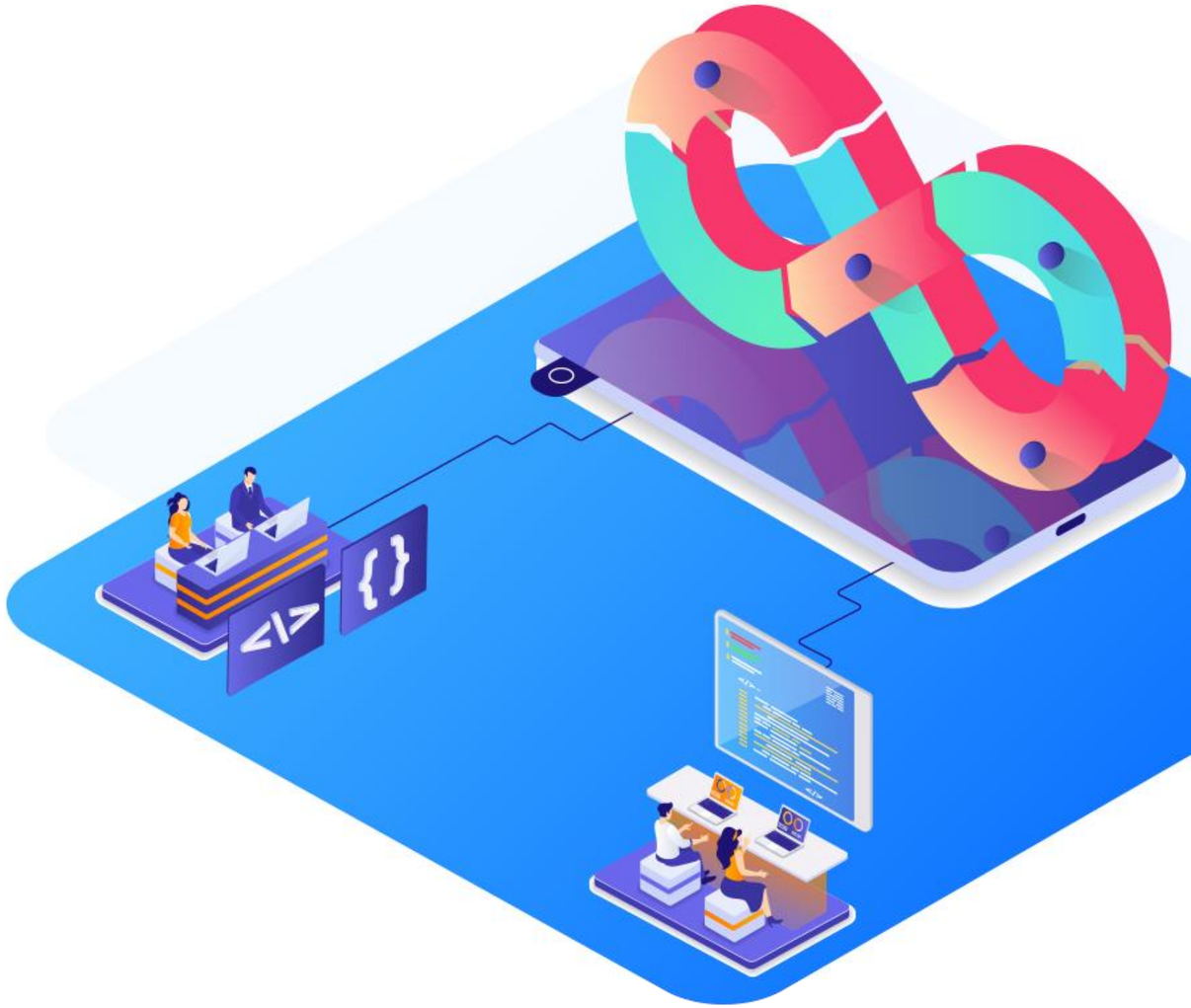
Configuration Management with Ansible and Terraform



Working with YAML

Learning Objectives

By the end of this lesson, you will be able to:

- Compare YAML with other data formats like JSON and XML to understand its unique features and benefits
- Classify different data types in YAML, including scalars, sequences, and mappings, to enhance data organization
- Implement anchors and aliases in YAML for reducing redundancy and maintaining consistency in configuration files
- Create YAML configuration files for various applications, ensuring readability and maintainability
- Utilize YAML syntax effectively to define hierarchical data structures and manage complex configurations

Introduction to YAML

What Is Yet Another Markup Language (YAML)?

It is an easy-to-read data serialization language often used to create configuration files with various programming language.

YAML uses indentation to define the file structure, which makes the code easier to write and understand.

Purpose of Using YAML

Following are the key purposes of using YAML:

Human-readable
YAML files are easy for humans to read and write due to its simple syntax.

Whitespace indentation
It uses indentation for structure, like Python, making it visually clear.

Expressiveness
It supports lists and dictionaries, suitable for diverse data representations.

Support for comments
It allows adding comments for context or explanations within the file.

Purpose of Using YAML

Precise error handling
It helps the user in quickly finding and fixing errors.

Support for complex structures
It helps the user in referring to other data objects by supporting complex data structures.

Easy integration
It seamlessly integrates with version control systems and configuration management tools.

Cross-language support
It supports many programming languages, which makes it versatile for different development environments.

Where to Use YAML?

Configuring files

YAML is used to set up and adjust settings in applications due to its readability and user-friendly format.

Automation scripts

It is utilized in automation scripts and configuration management tools to define workflows and specify infrastructure settings.

Cloud infrastructure

It is employed to define Infrastructure as Code (IaC) in cloud environments, specifying resources and configurations effectively.

Data sharing

It facilitates sharing data between diverse programming languages and systems with varying data structures, ensuring seamless interoperability.

Comparison with Other Data Formats

Comparing features of YAML alongside trending data formats like XML and JSON:

Aspect	YAML	XML	JSON
Definition	A readable data serialization language	A markup language for structured data	A lightweight data interchange format
Readability	Highly readable with indentation	Requires explicit tags, less readable	Readable, but less than YAML
Human friendliness	Designed for human readability	Designed primarily for machine readability	Designed primarily for machine readability
Flexibility	Supports complex data structures	Limited flexibility, rigid hierarchy	Supports basic data structures
Usage	Widely used in configuration files and Ansible playbooks	Commonly used in markup languages and web services	Commonly used for data interchange

YAML, XML, and JSON Syntax: Example

Example: Consider a list of users with their name and age written in different formats.

Explanation: This example demonstrate how the same data is structured in each format, illustrating the readability and conciseness of YAML compared to XML and JSON.

YAML

```yaml
users:
  - name: John
    age: 30
  - name: Jane
    age: 25
```

XML

```xml
<users>
  <user>
    <name>John</name>
    <age>30</age>
  </user>
  <user>
    <name>Jane</name>
    <age>25</age>
  </user>
</users>
```

JSON

```json
{
  "users": [
    {
      "name": "John",
      "age": 30
    },
    {
      "name": "Jane",
      "age": 25
    }
  ]
}
```

Quick Check

Which of the following is a feature of YAML that makes it suitable for IaC?

A. High Readability

B. Support for complex data structures

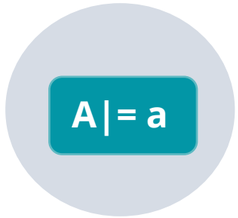
C. Diverse data repesentations

D. All of the above

Exploring YAML Syntax

Basic Rules for YAML Syntax

It follows strict rules to avoid confusion and ensure consistent, reliable usage across different languages and editors.



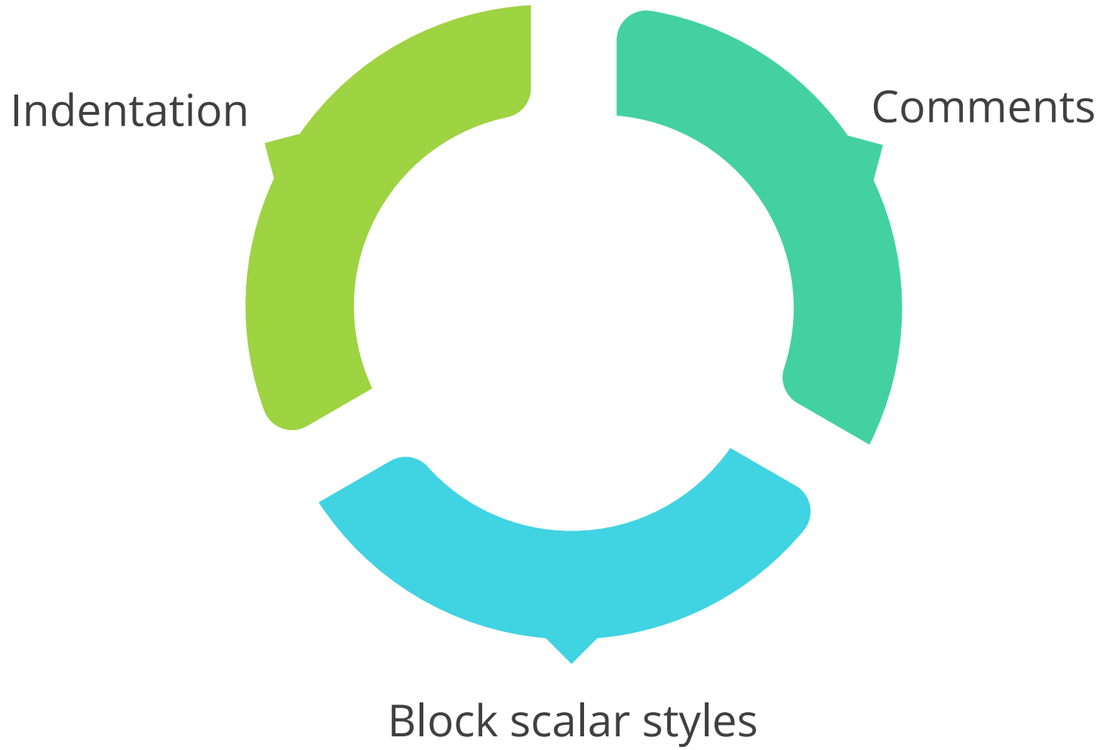
YAML is case-sensitive.

YAML files are saved with **.yml** or **.yaml** extension.

YAML does not allow the use of tabs while creating YAML files; however, spaces are allowed.

Key Concepts in YAML Syntax

To create and manage YAML files effectively, understanding the basics of YAML syntax is essential. Following are three fundamental aspects of YAML syntax:



YAML Syntax: Indentation

Indentation refers to the spaces or tabs added at the beginning of a line of text to visually indicate the level of nesting or hierarchy within a document or code.

Whitespace and indentation

YAML relies on whitespace and indentation to indicate nesting. The hierarchy and nesting are visible through a Python-like indentation style.

No tabs

Tab characters cannot be used for indentation in YAML files; only spaces can be used.

Consistency

The number of spaces used for indentation does not matter if they are consistent throughout the file.

YAML Syntax: Indentation

The following example shows the importance of consistent indentation in YAML:

```
tutorial:  # nesting level 1
  - yaml:  # nesting level 2 (2 spaces used for indentation)
      name: "YAML" # nesting level 3 (4 spaces used for indentation)
      type: awesome
      born: 2001
```

YAML Syntax: Comments

They are annotations or notes added within the YAML file to provide additional context, explanations, or remarks about the data or configuration being defined.

Comments are ignored by YAML parsers and have no impact on the data itself. They are preceded by the **#** character and extended until the end of the line.

YAML Syntax: Comments

The following code explains the use of comments in YAML files with the example of showcasing the company name:

```
# Comments inside a YAML file can be added followed by the # character
company: spacelift
```

YAML Syntax: Block Scalar Styles

For multiline strings, YAML provides block scalar styles. There are two types of block scalars:

1. Block literal style
2. Block folded style

Block Literal Style

It preserves newlines and leading whitespaces, allowing for a clean representation of multiline strings. In this example, the **|** denotes block literal style, and the content following **|** is treated as a multiline string.

Example:

```
data: |

 Every one

 Of these

 Newlines

 Will be

 Broken up.
```

Output:

```
Every one
Of these
Newlines
Will be
Broken up.
```

Block Folded Style

It is similar to block literal style, but it folds newlines into spaces unless a newline is followed by an empty or indented line. In this example, the **>** denotes the block folded style. The content following **>** is treated as a multiline string.

Example:

```
data: >
 This text
 is wrapped
 and will
 be formed into
 a single paragraph.
```

Output:

```
This text is wrapped and will be formed into a single paragraph.
```

What Does a Basic YAML Script Look Like?

Here is an example of a simple YAML script for employee records that follows the syntax rules:

```
---
#Employee Information

Employee-Id: 105672

Full-Name: Alice Smith

Department: Human Resources

Address:
 - street: 25 Oak Avenue
   city: New York
   state: NY

Position: Senior HR Specialist

Experience: |
  8 years in HR roles
  Certified in HR Management
...
```

Quick Check

Which of the following statements is incorrect about YAML?

A. YAML files are saved with a .yml extension.

B. YAML has built-in support for comments.

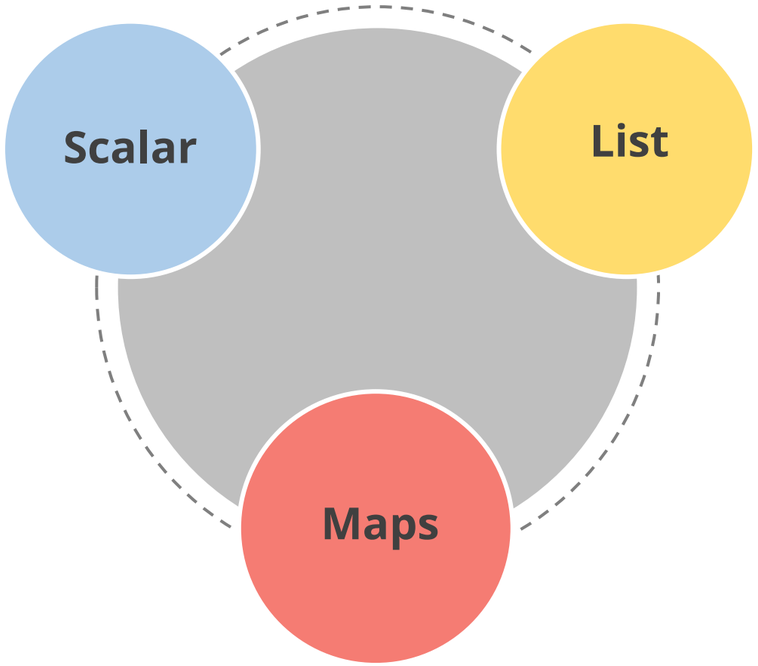
C. Tabs are used for indentation in YAML.

D. YAML is case-sensitive.

Data Types in YAML

Major Data Types in YAML

A single value, like a word or number

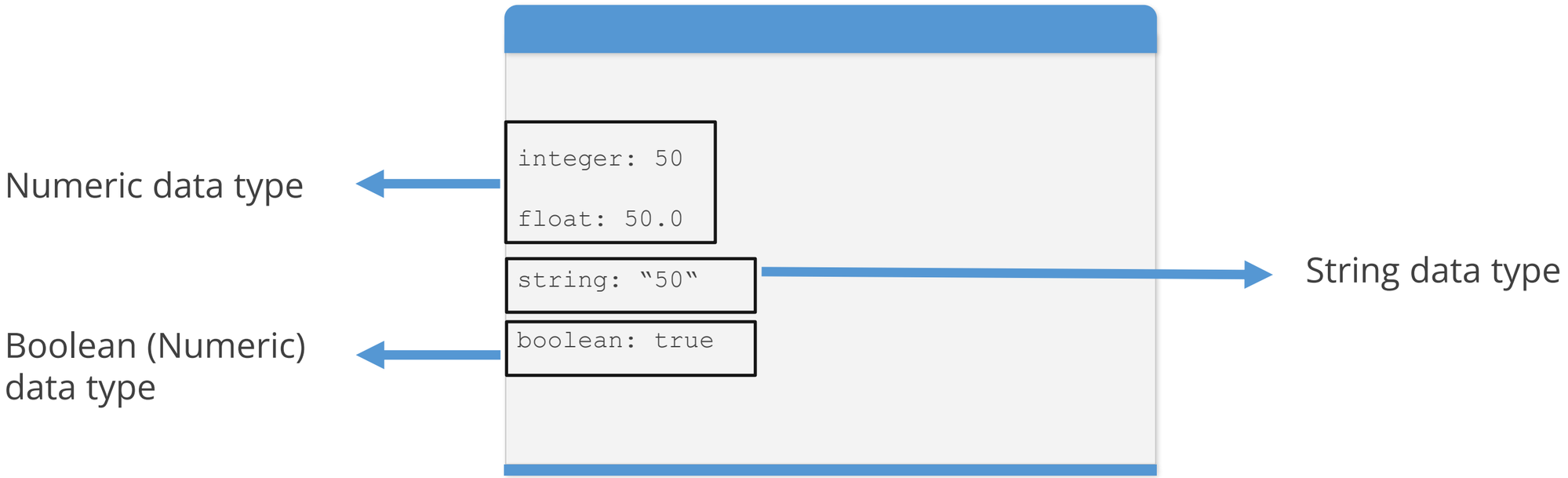


An ordered sequence of values, beginning with hyphens

An unordered collection of key-value pairs for organized data representation

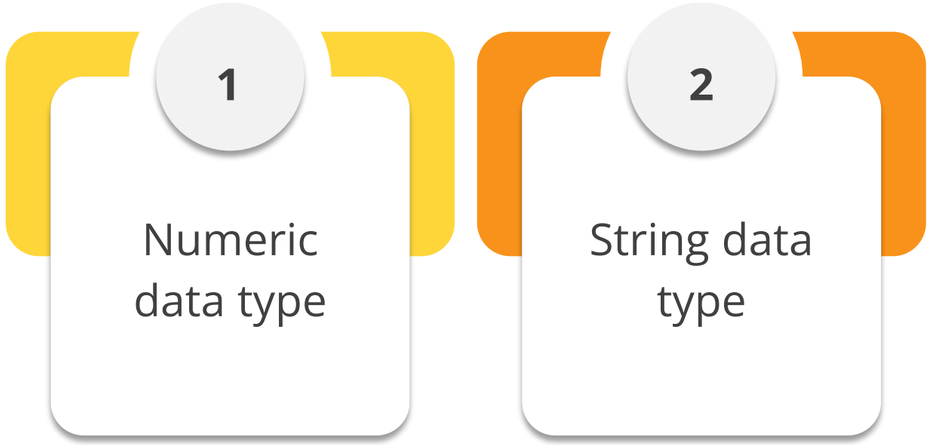
Scalars in YAML

It represents data as a series of Unicode characters. The following example represents scalar using different data types like strings, numeric, and Booleans:



Types of Scalars in YAML

Scalar data types are classified into two types:



Scalars in YAML: Numeric Data Type

The following are examples of numeric data types:

Integer and Floating

```
---

age: 25

binary: 11001

hex: 0x19

floating: 3.14

exp: 2.5e+03
```

Boolean

```
---

booleanvalue3: False

booleanvalue4: True

state1: Open

state2: Closed
```

Scalars in YAML: String Data Type

The following is an example of how YAML represents strings, including how it handles null values:

String Data Type

```
---

string: this is a string

str: "the cost is 390\n"

nullstr1: null

nullstr2: ~
```

Lists in YAML

It is a collection of values in a specific order that can contain any number of entries.

- A list starts with a dash (-) and a space.
- An indentation separates it from the parent.
- A list can be embedded into a map.

Lists in YAML

The following example showcases how a simple list is structured:

```
fruits:
  - Apple
  - Banana
  - Cherry
```

Lists in YAML

Here is how lists can be represented in YAML using both single-line and multi-line formats, as well as nested lists within complex objects:

List

```
---

#List in a single line
items: [6, 7, 8, 9, 10]
name: [six, seven, eight]

# List in multiple line
items:
  - 6
  - 7
  - 8

#complex objects
newitems:
 - values:
    value1:
    value2:
    value3:
```

Maps in YAML

It represents an unordered set of key-value pairs. The following example showcases how key-value pairs are structured within maps:

```
person:
  name: John
  age: 30
  city: New York
```

Each key within a map must be unique. YAML imposes no further restrictions.

Maps in YAML

Here is an example of how maps are structured in YAML, illustrating the use of key-value pairs to store employee data:

Maps

```
---

- developer1:
  empname: john
  skills:
   - python
   - java
   - yaml

- developer2:
  empname: martin
  skills:
   - C
   - perl
   - ruby
```

Quick Check

During a code review, you notice a junior engineer has incorrectly used YAML data types. You need to help them understand the basic scalar data types used in YAML. Which of the following represents a scalar data type in YAML?

A. String

B. List

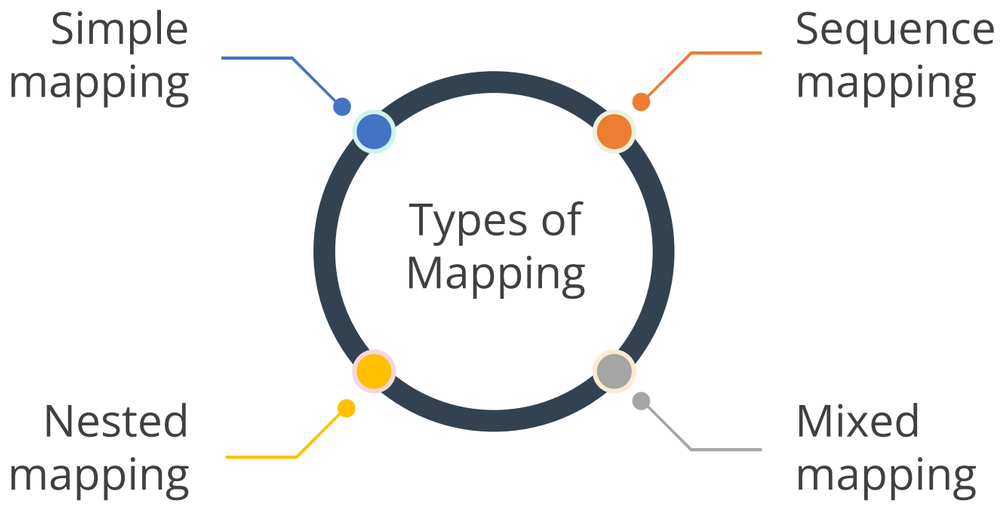
C. Dictionary

D. Set

Mapping and Sequencing in YAML

Mapping in YAML

It is a collection of key-value pairs, where each key is unique within the mapping, and each key is associated with a value. The following are various types of mapping:

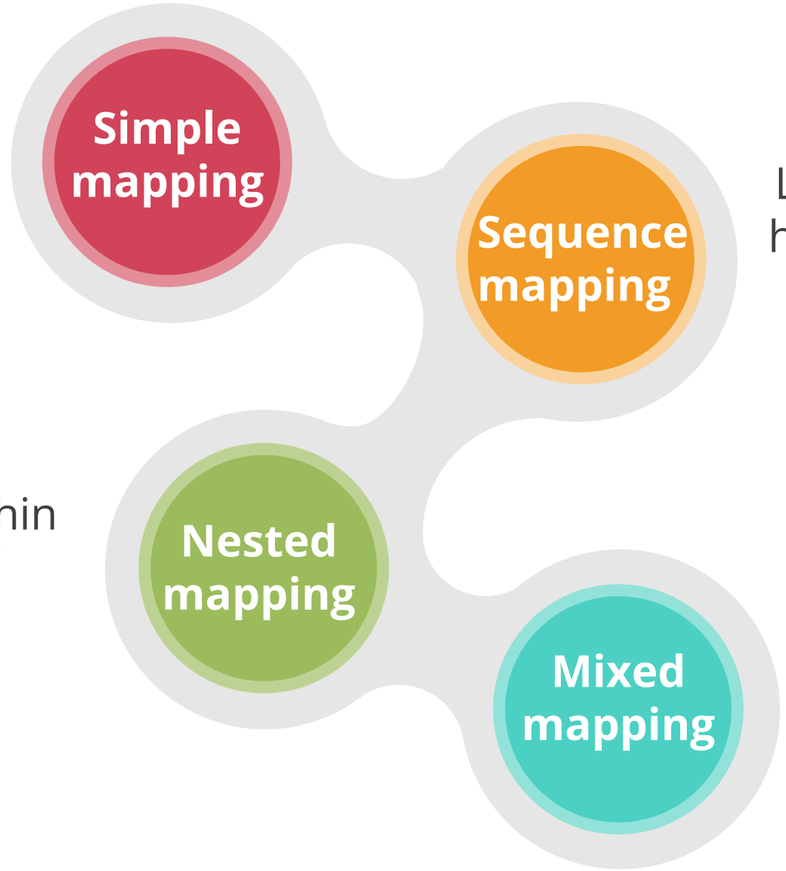


Note

In mapping, the names of two keys cannot be the same.

Types of Mapping

Key-value pairs represented by a colon in YAML, like **key: value**



Lists of items denoted by hyphens in YAML, such as **- item1** and **- item2**

Placing one mapping structure within another, creating a hierarchy of key-value pairs

Combination of simple, sequence, and nested mappings in YAML documents

Types of Mapping

Below are the examples of simple and sequence mapping:

Simple Mapping

```
---

hosts: xyz

emp: menon

job: developer
```

Sequence Mapping

```
---

student: charles

subjects:

 - maths

 - science
```

Types of Mapping

Below are the examples of nested and mixed mapping:

Nested Mapping

```
---

hosts: xyz

name: john

 address: canada

 lane: 13

 street: 42
```

Mixed Mapping

```
---

student: chris

details:
  fatherName: adam
  motherName: liam

subjects:
    - subject1
    - subject2
```

Differences Between Map and Mapping

Map

- It is an unordered collection of unique key-value pairs, like dictionaries in Python or hash tables in other languages.

Mapping

- It refers to the specific syntax used in YAML to represent a map.
- The term **mapping** is often used when discussing the YAML syntax and the way maps are written in YAML files.

Sequencing in YAML

A sequence in YAML is a collection of related items, where the order of items is preserved.

```
---
hobbies:
    - dancing
    - music
    - singing
    - painting
```

Sequencing in YAML: Nested Sequence

It is a sequence that is contained within another sequence. Items and subitems in a nested sequence can be achieved by placing a single space before each dash in the subitems.

Nested Sequence

```
---

hosts: xyz

name: john

 address: canada

 lane: 13

 street: 42

skills:

 - python

 - java
```

Assisted Practice

Creating, validating, and parsing a YAML file using Python **Duration: 10 Min.**

Problem Statement:

You've been assigned a task to create, validate, and read a YAML file using Python for managing configurations or data serialization tasks, ensuring the file is read and validated correctly.

Assisted Practice: Guidelines

Steps to be followed:

1. Create a YAML file using Python
2. Run the Python file

ASSISTED PRACTICE

Assisted Practice

Configuring Apache web server using Ansible **Duration: 10 Min.**

Problem Statement:

You've been assigned a task to configure and validate the setup of the Apache web server using Ansible on a local node machine for automated server management.

Assisted Practice: Guidelines

Steps to be followed:

1. Establish connectivity between the Ansible controller and node machine
2. Configure and validate the Apache web server setup

Quick Check

You are designing a complex configuration for a multi-tier application. A team member asks for clarification on how to properly use mappings and sequences in YAML.
What is the primary difference between a mapping and a sequence in YAML?

A. Mapping represents a list of items, while sequence represents key-value pairs.

B. Mapping and sequence are interchangeable terms in YAML.

C. Mapping and sequence both represent hierarchical structures in YAML.

D. Mapping represents key-value pairs, while sequence represents a list of items.

Anchors, Aliases, and Tags

Anchors (&) in YAML

It allows users to define reusable content that can be referenced multiple times within the document.

```
# Define YAML Anchors
default_settings: &defaults
  adapter: mysql
  host: localhost
```

&default_settings defines an anchor named **default_settings.**

Aliases (*) in YAML

It references previously defined anchors, ensuring consistent configurations.
Overrides **(<<:)** are used to merge values.

```
development:
  <<: *defaults
  database: dev_db

test:
  <<: *defaults
  database: test_db
```

***defaults** references the anchor **defaults.**

<<: *defaults merges the values from the **defaults** anchor into the current section.

References in YAML Configuration

They are a way to avoid redundancy by defining an alias (reference) and reusing it multiple times within the document. This helps in maintaining a cleaner, more readable, and more maintainable YAML file.

YAML uses **anchors (&)** and **aliases (*)** to implement references.

When to Use References?

The following are the key scenarios where using references in YAML can be beneficial:

01 When there are multiple sections in a YAML file that share the same structure or data

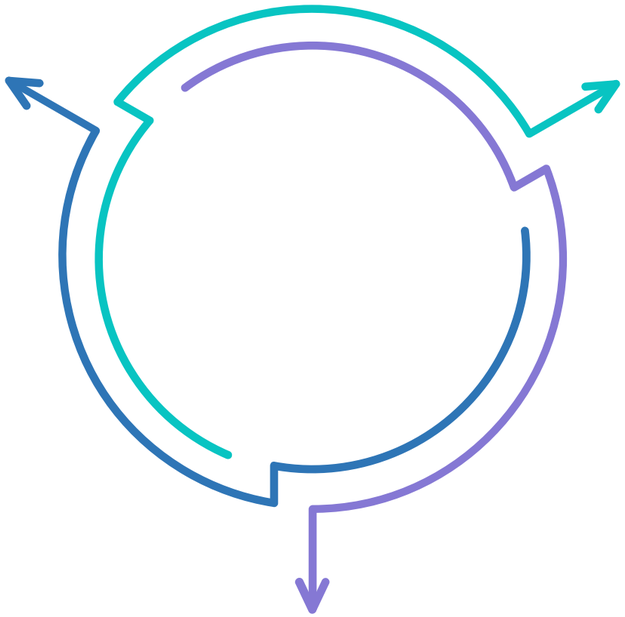
02 When a user wants to keep the YAML file DRY (Don't Repeat Yourself)

03 When a user needs to maintain consistency across similar data entries

What Are YAML Tags?

They are simple identifiers used to represent the type of native data structures. Tags can be used to:

Set a custom URI (Universal Resource Indicator)



Set local tags within the YAML file

Set the data type of a variable

Setting a Custom URI

It defines unique identifiers for specific data structures. Custom URIs must be defined before the document starts (before **---**).

Syntax:

```
%TAG ! tag:prefix
---
```

Example:

```
%TAG ! tag:custom-uri
---
server: Linux-Server
datacenter:
    location: new-york
    zones: 12
    points: 34
roles:
    - backup
```

Setting a Local Tag

In this example, tags are specific to the YAML file for local context. It is declared using **!** and a prefix, specific to the YAML file.

Syntax:

```
name: ! local-tag-name value
```

Example:

```
%TAG ! tag:custom-uri
---
server: Linux-Server
datacenter:
    location: ! LOC new-york # local tag
    zones: 12
    points: 34
roles:
    - backup
```

Setting Data Type in YAML

In this example, tags are used to explicitly declare the type of data (e.g., integer and string).

Syntax:

```
!!datatype
```

Example: Specify zip as a string

```
house:
  road: "main road"
  city: "Big City"
  zip: !!str 12345
  floor: 4
```

Assisted Practice

Implementing YAML anchors and aliases **Duration: 10 Min.**

Problem Statement:

You've been assigned a task to implement the YAML anchors and aliases in Ansible to efficiently manage and reuse repeated tasks, reducing redundancy and enhancing maintainability in playbooks.

Assisted Practice: Guidelines

Steps to be followed:

1. Create Ansible playbook using YAML anchors and aliases
2. Run the Ansible playbook

ASSISTED PRACTICE

Assisted Practice

Implementing global variables in Ansible **Duration: 10 Min.**

Problem Statement:

You've been assigned a task to implement global variables in Ansible, define them centrally, and reference them in playbook tasks ensuring consistent variable usage across tasks and hosts

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed:

1. Set up the directory structure
2. Create a global variables file
3. Create a playbook
4. Run the playbook

Quick Check

You are managing configurations for a web application. You notice that many configuration settings are repeated across these environments. How can you avoid redundancy and ensure consistency across your configuration files?

A. Use different keys for each environment

B. Copy-paste the configurations for each environment

C. Define reusable configuration blocks with anchors and reference them using aliases

D. Write separate YAML files for each environment without sharing common configurations

Working with YAML Files

YAML File Operations

The following are the various operations for working with YAML files:

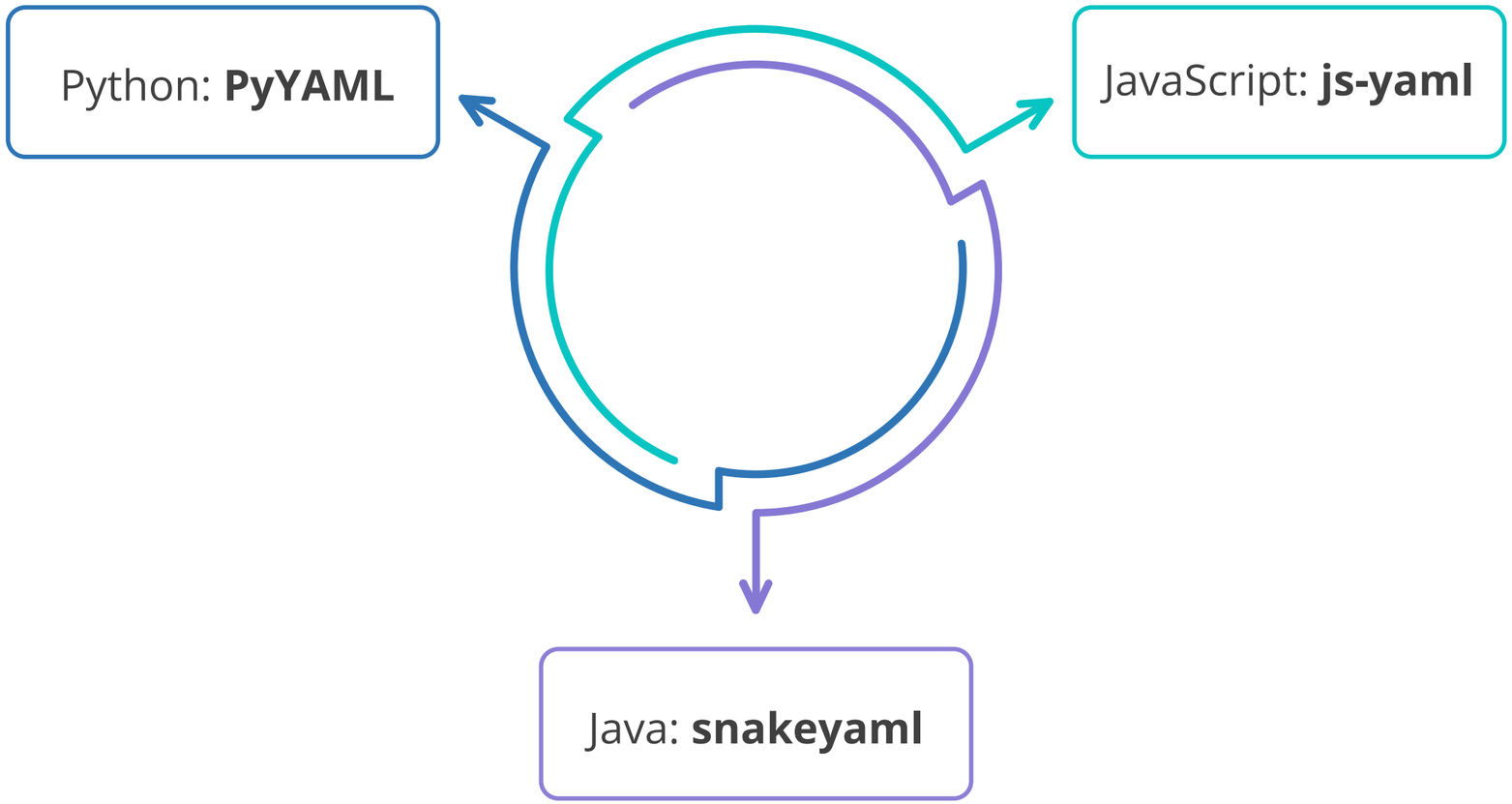
Loading YAML data

Saving YAML data

Validation and error handling

Loading YAML Data

It means reading YAML files and parsing them into a usable format within a program. The following are some common libraries used to load YAML data:



Loading YAML in Python

The following is an example showing how you can load YAML data in a Python program:

Example:

```
import yaml  # Import the PyYAML library

# Open the YAML file and load its contents
with open('data.yaml', 'r') as file:
    data = yaml.safe_load(file)

print(data)  # Print the loaded data
```

yaml.safe_load(file): safely loads the YAML file content into a Python dictionary.

Loading YAML in JavaScript

The following is an example showing how you can load YAML data in a JavaScript application:

Example:

```
const yaml = require('js-yaml');  // Import the js-yaml library
const fs = require('fs');  // Import the file system module

try {
  // Read the YAML file and parse its contents
  const doc = yaml.load(fs.readFileSync('data.yaml', 'utf8'));
  console.log(doc);  // Print the loaded data
} catch (e) {
  console.log(e);  // Handle any errors
}
```

yaml.load(fs.readFileSync('data.yaml', 'utf8')): reads and parses the YAML file content.

Loading YAML in Java

The following is an example showing how one can load YAML data in a Java application:

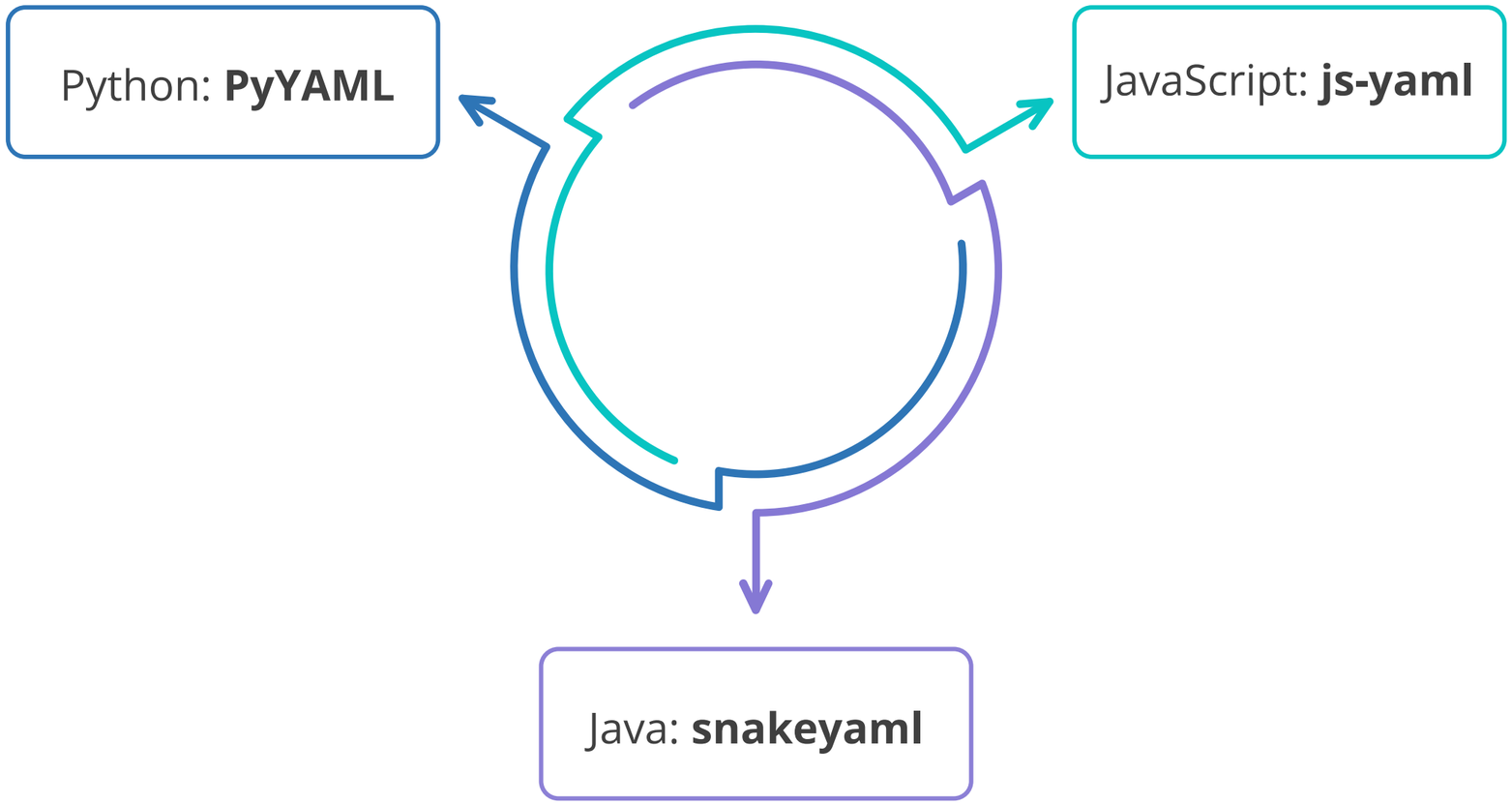
Example:

```
import org.yaml.snakeyaml.Yaml;  // Import the SnakeYAML library
import java.io.InputStream;
import java.util.Map;

public class Main {
    public static void main(String[] args) {
        Yaml rags = new Yaml();  // Create a new Yaml instance
        InputStream inputStream = Main.class
            .getClassLoader()
            .getResourceAsStream("data.yaml");

        // Load the YAML file and parse its contents into a Map
        Map<String, Object> data = yaml.load(inputStream);
        System.out.println(data);  // Print the loaded data
    }
}
```

yaml.load(inputStream): loads and parses the YAML file content into a Java Map.

Saving YAML Data

It involves converting data structures into YAML format and writing them into a file. The following are some common libraries used to save YAML data:



Saving YAML in Python

The following is an example showing how a user can save data to a YAML file in Python:

Example:

```
import yaml  # Import the PyYAML library

# Define the data to be saved
data = {'name': 'John', 'age': 30, 'city': 'New York'}

# Open a file and save the data to it in YAML format
with open('data.yaml', 'w') as file:
    yaml.dump(data, file)
```

yaml.dump(data, file): converts the Python dictionary to YAML format and writes it into a file.

Saving YAML in JavaScript

The following is an example that demonstrates how to save data to a YAML file in JavaScript:

Example:

```
const yaml = require('js-yaml');  // Import the js-yaml library
const fs = require('fs');  // Import the file system module

// Define the data to be saved
const data = { name: 'John', age: 30, city: 'New York' };

try {
  // Write the data to a file in YAML format
  fs.writeFileSync('data.yaml', yaml.dump(data), 'utf8');
} catch (e) {
  console.log(e);  // Handle any errors
}
```

yaml.dump(data): converts the JavaScript object into YAML format.

fs.writeFileSync('data.yaml', yaml.dump(data), 'utf8'): writes the YAML data into a file.

Saving YAML in Java

The following code demonstrates how to save data to a YAML file in Java:

Example:

```
import org.yaml.snakeyaml.Yaml;  // Import the SnakeYAML library
import java.util.Map;  // Import the Map interface
import java.io.FileWriter;  // Import FileWriter for writing files

public class Main {
    public static void main(String[] args) throws Exception {
        Yaml File Writer = new Yaml();  // Create a new Yaml instance

        // Define the data to be saved
        Map<String, Object> data = Map.of("name", "John", "age", 30, "city", "New
York");

        // Dump data to the YAML file
        yaml.dump(data, new FileWriter("data.yaml"));  // Write the data to a YAML file
    }
}
```

yaml.dump(data, new FileWriter("data.yaml")): converts the Java Map to YAML format and writes into to a file.

Validation and Error Handling in YAML

This ensures data integrity and manages errors effectively when working with YAML files.

Importance of Validation

The following are the key reasons why validation is important:

Consistency

Validation ensures that the YAML data conforms to the expected structure and schema.

Accuracy

It helps in catching data format errors early in the process.

Security

It prevents malicious data from causing vulnerabilities in the application.

Error Handling Techniques

Try-catch blocks

The following example shows how users can use try-catch blocks with Python to manage errors during YAML file operations:

Python

```
import yaml

try:
    with open('data.yaml', 'r') as file:
        data = yaml.safe_load(file)
except yaml.YAMLError as e:
    print(f"Error loading YAML file: {e}")
```

Error Handling Techniques

Try-catch blocks

The following example shows how users can use try-catch blocks with JavaScript to manage errors during YAML file operations:

JavaScript

```
const yaml = require('js-yaml');
const fs = require('fs');

try {
  const doc = yaml.load(fs.readFileSync('data.yaml', 'utf8'));
} catch (e) {
  console.error(`Error loading YAML file: ${e}`);
}
```

Error Handling Techniques

Logging

Log errors to help with debugging and monitoring the application

User feedback

Provide clear and meaningful error messages to users for better user experience

Best Practices for Writing YAML Files

The following are some guidelines to ensure high-quality and maintainable YAML files:

01 Always use consistent indentation, preferably spaces over tabs, to avoid parsing errors

02 Give the keys descriptive and meaningful names for better readability

03 Keep nesting levels to a minimum to maintain clarity and simplicity

Best Practices for Writing YAML Files

04 Remember that YAML is case sensitive; therefore, be consistent with the case used in your keys

05 Use comments to document your YAML files wherever necessary

06 Separate sections with blank lines to enhance readability between different configurations

Quick Check

You need to read a YAML configuration file into your Python application to dynamically load settings for different environments. Which Python library and method would you use to safely load and parse the YAML file?

A. json and json.load()

B. PyYAML and yaml.safe_load()

C. yaml2json and yaml2json.load()

D. yamlparser and yamlparser.read()

YAML in Different Contexts

YAML as Configuration Files

Configuration files are used to set parameters and initial settings for some computer programs.

They enable the software to adjust its operation according to the user's preferences or specific environments.

YAML as Configuration Files

The following is an example of a YAML configuration file:

Example:

```yaml
server:
  port: 8080
  host: localhost

database:
  type: mysql
  host: 127.0.0.1
  port: 3306
  username: root
  password: password

features:
  enableFeatureX: true
  enableFeatureY: false
```

YAML is human-readable and easy to read and write, even for non-programmers. It supports hierarchical structures, making it ideal for nested configurations.

YAML for Data Serialization

Data serialization is the process of converting data structures or object states into a format that can be stored or transmitted and reconstructed later.

YAML is flexible and supports complex data types and structures, making it suitable for a wide range of serialization needs.

YAML for Data Serialization

The following is an example of data serialization with YAML:

Example:

```
person:
  name: John Doe
  age: 30
  address:
    street: 123 Main St
    city: Anytown
    state: CA
    zip: 12345
  hobbies:
    - reading
    - cycling
    - hiking
```

Assisted Practice

Using YAML in Python for data serialization **Duration: 5 Min.**

Problem Statement:

You've been assigned a task to demonstrate how to use YAML in Python for data serialization, making data more readable, and easier to manage.

Assisted Practice: Guidelines

Steps to be followed:

1. Create a Python script for data serialization using YAML
2. Run the serialization script

ASSISTED PRACTICE

Quick Check

You are setting up configuration files for a web server that needs to run in various environments. These files should be easy for your team to read and update. Why is YAML a good choice for writing these configuration files?

A. YAML files are binary and difficult to read.

B. YAML supports complex data structures and is human-readable.

C. YAML does not support comments, making it less cluttered.

D. YAML files are encrypted by default.

Key Takeaways

- YAML is an easy-to-read data serialization language often used to create configuration files with any programming language.
- Scalars represent data as a series of Unicode characters.
- A list is a collection of values in a specific order that can contain any number of entries.
- Maps represent an unordered set of key-value pairs.
- A sequence in YAML is a collection of related items, where the order of items is preserved.

Key Takeaways

- Anchors allow users to define reusable content that can be referenced multiple times within the document.
- Aliases reference previously defined anchors, ensuring consistent configurations.
- YAML is human-readable and easy to read and write, even for non-programmers. It supports hierarchical structures, making it ideal for nested configurations.

Installing Apache Tomcat Using Ansible Playbook

Duration: 25 Min.

Project agenda: To install Apache Tomcat using an Ansible playbook for automated deployment, configuration, and management of the application server

Description: As a DevOps engineer at FutureTech Solutions, you are tasked with automating the deployment of Apache Tomcat on multiple Ubuntu servers. Your objectives include creating an Ansible playbook for consistent installation and configuration, implementing Ansible Vault to manage sensitive data securely, and developing a dynamic inventory system to adapt to server changes, ensuring scalability and reducing manual intervention.

LESSON-END PROJECT

Installing Apache Tomcat Using Ansible Playbook

Duration: 25 Min.

Perform the following:

1. Run Ansible and update packages
2. Configure Ansible
3. Establish connectivity between the Ansible controller and the node machine
4. Create an Apache playbook
5. Execute the playbook
6. Confirm the installation

Expected deliverables: A step-by-step guide to install Tomcat, begins by writing a playbook that outlines all the necessary tasks for downloading, configuring, and starting the Tomcat service.

LESSON-END PROJECT

Thank You