YAML - MAPPING

Yaml Breakdown

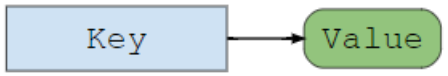
- **Mapping**
- Sequences
- Scalars
- Structures
- Comments
- Tags
- Anchors

Mapping

- Defining key-value pairs in YAML.
- Nested mappings.
- Examples and use cases.

What Is Key-value Pair?

- A key-value pair is a simple data structure that consists of a unique identifier (the key) and the corresponding value of that identifier.
- The **Key** can be any type of data, such as:
  - A Text String
  - An Integer
- The **Value** can also be of any type of data:
  - Including string
  - Integer
  - Float
  - Boolean
  - list, or even other key-value pairs.



A key-value pair

Uses Of Key-value Pairs

- Most common uses of key-value pairs:
  - They are used in the implementation of hash tables, whereby they are used to store and retrieve information.
  - Many programming languages include built-in support for key-value pairs as a core data structure, making them a fundamental part of many software applications.
- Overall, key-value pairs provide a powerful and flexible way to store and work with data.

Syntax

- In YAML, key-value pairs are represented using a colon (:), followed by the content of the value.
- The basic format of a key-value pair is as follows:
  - key: value
- Where key represents the key, and value represents the corresponding value.

Nested Mapping

- In a YAML file we can map key-value pairs inside each other this type of mapping is known as nested mapping.
- Example:

```
---
foo:
  nested_map:
    key: value

```

Use Cases Of Mapping (1-2):

- Configuration Settings:
  - Storing configuration settings for Ansible playbooks, roles, or tasks, such as server settings, API endpoints, or database configurations.
- User Profiles:
  - Defining user profiles with various attributes, useful for managing user-related operations in Ansible, like user provisioning or access control.
- Product Information:
  - Managing product details, prices, and specifications for use in Ansible playbooks related to product deployment or provisioning.

Use Cases Of Mapping (2-2):

- Environmental Configuration:
  - Storing environment-specific configurations, enabling flexible deployment and setup in different environments (e.g., production, staging, testing).
- Nested Configurations:
  - Organizing configurations hierarchically to represent complex relationships or nested data structures for infrastructure components.
- Menu or Navigation Structure:
  - Defining menu items or navigation structures for web applications or interfaces, useful for configuring the interface in Ansible-based deployments.

Examples

Configuration Files

```yaml
---
server:
  port: 8080
  host: example.com
database:
  name: my_database
  username: my_user
  password: my_password
```

User Profile

```yaml
---
server:
  port: 8080
  host: example.com
database:
  name: my_database
  username: my_user
  password: my_password
```

Environment Config File

```yaml
---
environment:
  production:
    database_url: prod.example.com/db
    api_url: api.example.com
  staging:
    database_url: staging.example.com/db
    api_url: api-staging.example.com
```

Nested File

```yaml
---
parent:
  child1:
    key1: value1
    key2: value2
  child2:
    key3: value3
```