Data Structures, YAML and JSON

Ivan Pepelnjak (ip@ipSpace.net)
Network Architect

ipSpace.net AG

Live demos

- YAML parsers in Perl and Python
- YAML-to-JSON converter in Python
- Running on Ubuntu 14.04 LTS
- Source code in ipSpace.net Github repository

github.com/ipspace/NetOpsWorkshop/tree/master/YAML

Introduction to YAML

What Is YAML?

YAML Ain't Markup Language (from yaml.org)

> YAML is a human friendly data serialization standard for all programming languages

- Similar to XML or JSON (but easier to read)
- Represents single values, lists or key-value pairs
- Language specification on yaml.org

YAML in real life:

- Libraries available in C, Ruby, Python, Java, Perl, PHP, JavaScript…
- Used as configuration format by numerous open-source tools (including Ansible)

Why YAML?

Why would you use text file-based configuration:

- Treat configurations (or network state) as source code
- Change with any text editor
- Use source-code repositories and versioning tools (Git, SVN, RCS…)
- Implement workflows with tools like Gerrit

Why would you use YAML:

- Easier to write than JSON or XML
- Line-oriented ➔ easier to diff

YAML Syntax

```
# This is a comment
# Three dashes start a new document
---
log_dir: logs
build_dir: build
domain_name: lab.ipspace.net
```

- Indentation matters
- A single YAML file can contain multiple documents
- Every document starts with three dashes
- Comments start with #

Introduction to JSON

What Is JSON?

JSON = JavaScript Object Notation (from json.org)

> JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate.

- Based on a subset of the JavaScript Programming Language (ECMA-262)
- Represents single values, lists or key-value pairs
- Language specification on yaml.org
- Libraries available in C, Ruby, Python, Java, Perl, PHP, JavaScript…
- Most common data interchange format (including Ansible)

JSON Syntax

```
1. Object :== { string:value,…}
2. Array  :== [ value, … ]
3. Value  :== String | Number | Object | Array |
              true | false | null
4. String :== "..."
```

- Extremely simple syntax
- Can be prettified for easy reading or minified for optimal bandwidth utilization
- Strings **must** use double quotes

Why Would You Use JSON and not YAML?

JSON advantages

- Easier to parse
- No line-break, whitespace or indentation requirements

JSON disadvantages:

- Harder to read and write (by humans) as compared to YAML
- Don't even try to read minified JSON
- No namespaces (like XML)

Minification: the process of removing all unnecessary characters from source code without changing its functionality.

Simple Data Types

Scalar Values

Scalar Value in YAML

```
# This is a comment
# Three dashes start a new document
---
SomeValue
```

- YAML document can contain a single value
- Strings don't have to be quoted unless they contain special characters (-, :, {, }…)
- Multiline strings start with | or > (more in a few slides)
- Don't use comments (#) in the same line as the scalar value

Sample Scalar Values

```yaml
# This is really a list of scalar values
---
- 12345         # Number
- True          # Boolean
- Brocade       # String
- http://www.cisco.com   # A URL is a string
- "Quoted string"
```

Dynamic typing (or not) and comment handling is library-dependent

Multi-line Values

```
# Some long strings
---
- |
  multi-line string
  indentation indicates it's still the same
  scalar value.
  !! Newlines are preserved !!
- >
  multi-line string rolled
  into a single line
```

Multiline string value must start with "|" or ">"

Scalar Value in JSON

```
"SomeValue"
```

- Strings are enclosed in double quotes
- Special characters (for example, double quotes) are escaped with \
- Numbers, **true**, **false** or **null** are printed verbatim

Simple Lists

Lists 101

- Ordered collection of values
- Accessed by absolute position of an item (zero or one-based)
- Each value could be a simple value, list, object…
- In most modern languages the values could have different types

Known as

- Arrays (Pascal, C, Perl, JavaScript, JSON)
- Lists or arrays (Python)
- Sequences (YAML)

0	**123**
1	**abcd**
2	**456**
3	**def**
4	**foobar**

YAML Sequences (Lists)

```yaml
# List of some major network vendors
---
- Juniper
- Cisco
- Brocade
- F5     # Load balancer is in the network
```

Simple YAML lists

- One item per line
- Every item starts with a dash
- Value of the item is the rest of the line
- Comments are treated differently by various parsers

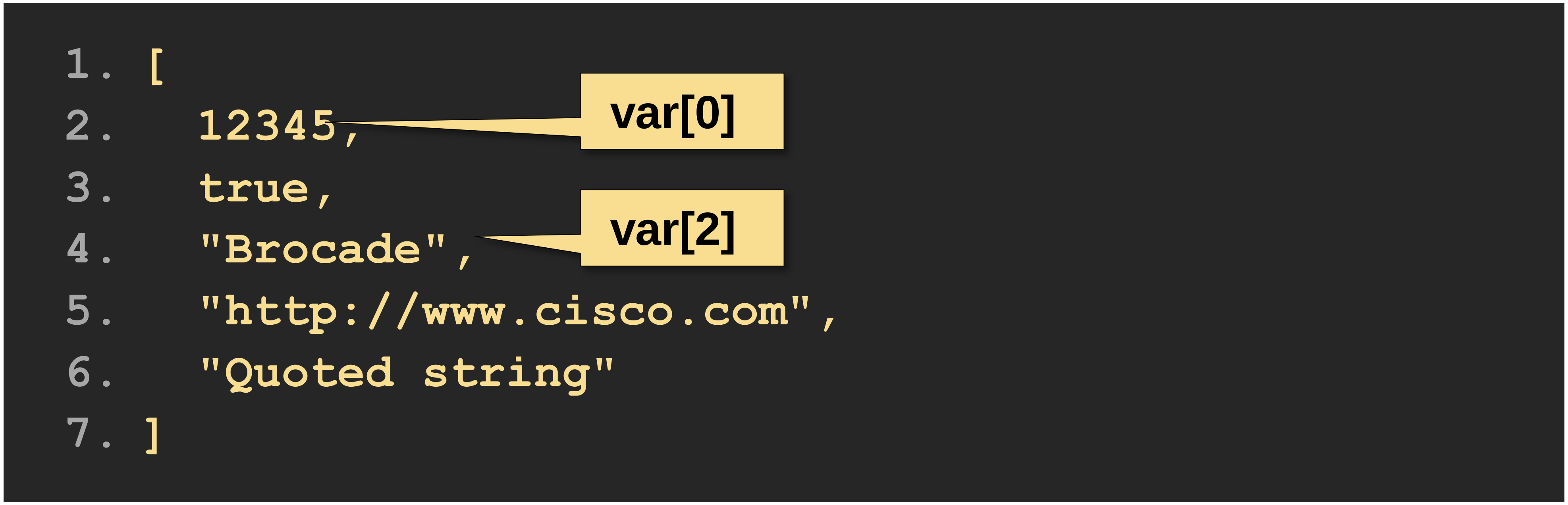
Lines with scalar values may or may not contain comments

JSON Arrays (Lists)

```
[
    12345,
    true,
    "Brocade",
    "http://www.cisco.com",
    "Quoted string"
]
```

- A list starts with **[** and ends with **]**
- List values are separated with commas
- Indentation and newlines don't matter

Accessing Arrays/Lists in Python or Jinja2



```
[
    12345,
    true,
    "Brocade",
    "http://www.cisco.com",
    "Quoted string"
]
```

- Square brackets indicate array reference
- First element in the array has index 0

Multi-line Values (JSON)

```
[
  "multi-line string\nindentation indicates…\n",
  "multi-line string rolled into a single line"
]
```

- JSON does not support multi-line values
- Everything between matching quotes becomes the string value (including line breaks)
- Line breaks are usually displayed as \n

Simple Key-Value Objects

Key-Value Objects 101

- Unordered collection of key-value pairs
- Accessed by key content
- Keys are scalar values (numbers or strings)
- Each value could be a scalar value, list, object…

Known as

- Object (JavaScript, JSON)
- Hash (Perl)
- Dictionary (Python)
- Mappings (YAML)

hostname	**R1**
ip	**1.2.3.4**
banner	**Don’t touch**
id	**12**
username	**foobar**

Key-Value Pairs (Hashes, Dictionaries) in YAML

```yaml
# Router description
---
hostname: R1
loopback_IP: 192.168.0.1
loopback_subnet: 255.255.255.255
banner: |
  The configuration is managed by Ansible.

  Don't change it - your changes will be lost.
```

- A YAML document can be a set of key-value pairs (variables, properties)
- Each value (property) can have a simple (scalar) or complex (list, dictionary) value

JSON Objects

```json
{
    "banner": "The configuration of this…",
    "hostname": "R1",
    "loopback_IP": "192.168.0.1",
    "loopback_subnet": "255.255.255.255"
}
```

- A JSON object starts with { and ends with }
- Keys are strings and **must** be quoted
- Colon separates key and value in a pair
- Values can be any valid JSON value
- Key-value pairs are separated by commas

Accessing Dictionary Values in Python and Jinja2

var.banner

```
{
   "banner": "The configuration of this…",
   "hostname": "R1",
   "loopback_IP": "192.168.0.1",
   "loopback_subnet": "255.255.255.255"
}
```

var["loopback_subnet"]

Python

- Dictionaries are accessed using array notation (var[key])

Jinja2

- Templates can use dot notation when the key is a valid Python variable name (ASCII letters, numbers, underscore)
- Array notation must be used otherwise

Complex Data Types

Complex Data Types

Value in a list or dictionary can be another list or dictionary

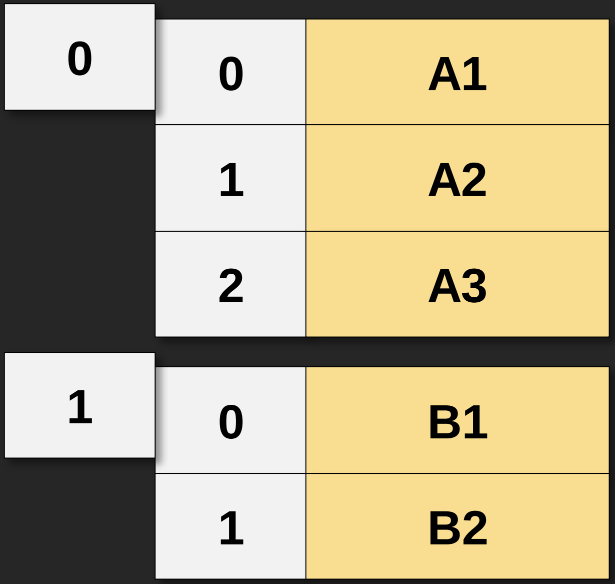
Sample data structures

- Lists of lists
- Lists of dictionaries
- Dictionary of lists and dictionaries

Lists of Lists

Complex Data: Lists of Lists (YAML)

```
# 2x3 table
---
- - Cell A1
  - Cell A2
  - Cell A3
- - Cell B1
  - Cell B2
  - Cell B3
```



- A value of a list item is another list (rarely used)
- The second list is also started with a dash
- Indents are used to indicate hierarchy

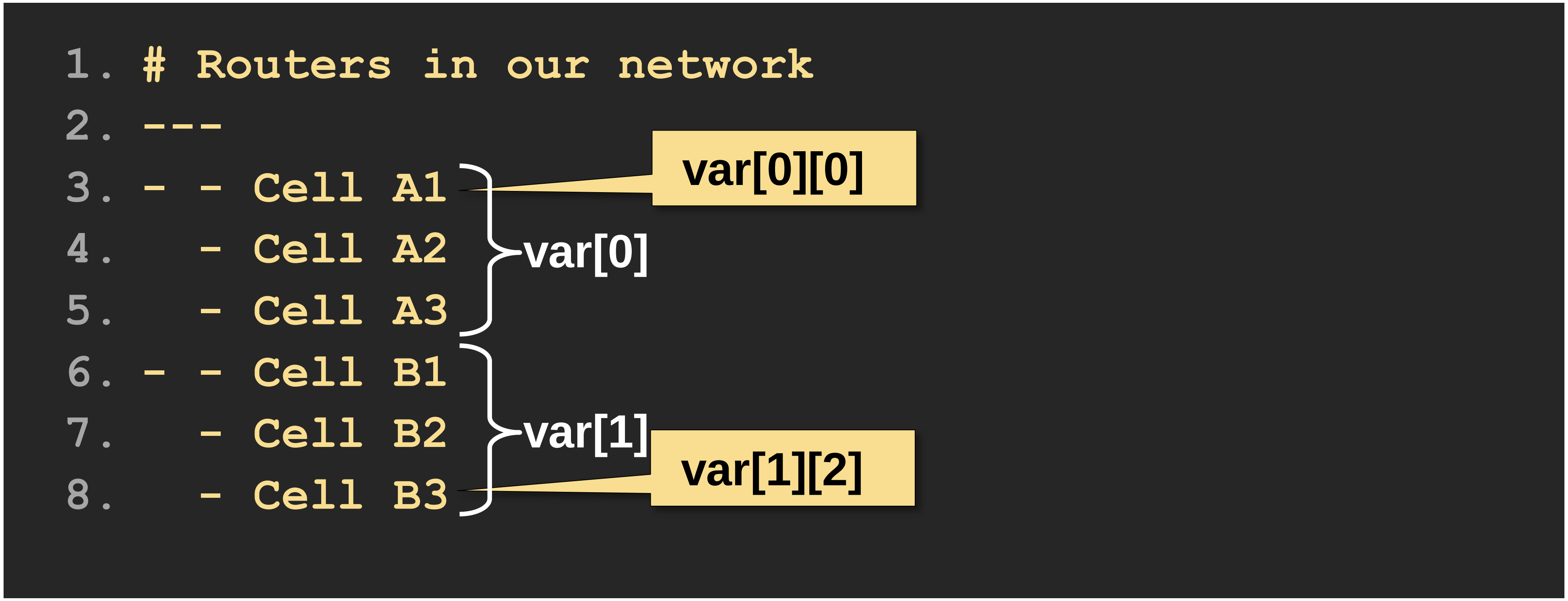
Complex Data: Lists of Lists (JSON)

```
[
  [
    "Cell A1",
    "Cell A2",
    "Cell A3"
  ],
  [
    "Cell B1",
    "Cell B2",
    "Cell B3"
  ]
]
```

Inner list

- Second [starts the inner list

Referencing Lists of Lists Elements in Python/Jinja2



```
# Routers in our network
---
- - Cell A1
  - Cell A2
  - Cell A3
- - Cell B1
  - Cell B2
  - Cell B3
```

- First array reference selects an element in the outer list
- Selected element is a list
- Second array reference selects an element in the inner list

Lists of Dictionaries

Complex Data: List of Dictionaries (YAML)

```
# Routers in our network
---
- description: DMVPN routers
- hostname: R1
  loopback: 192.168.0.1
- hostname: R2
  loopback: 192.168.0.2
- hostname: R3
  loopback: 192.168.0.3
```

dictionary

dictionary

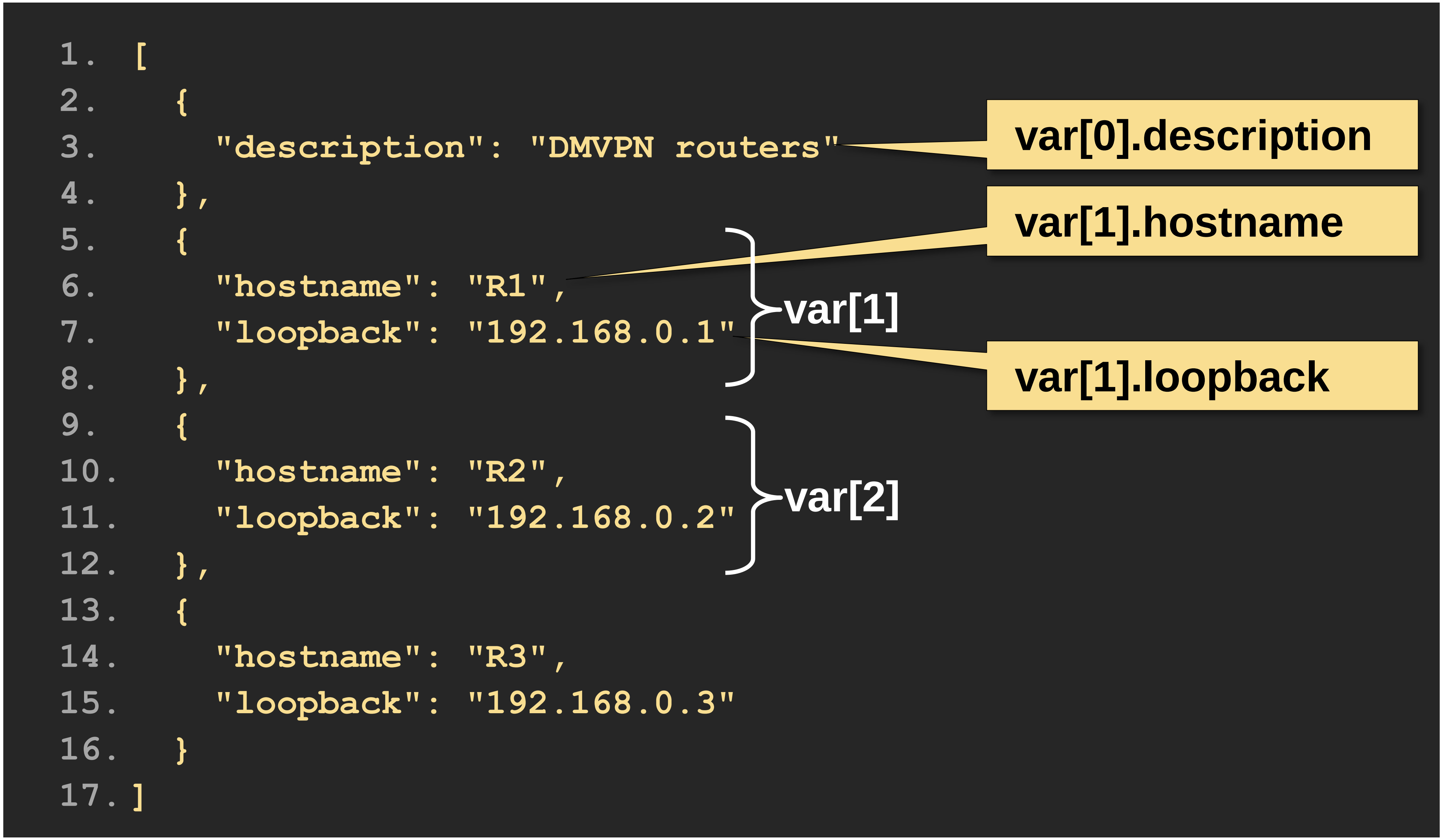
- A value of a list item can be another list or dictionary
- Indentation indicates the hierarchy

First dictionary key follows the dash, second key is aligned with the first

Complex Data: List of Dictionaries (JSON)

```
[
  {
    "description": "DMVPN routers"
  },
  {
    "hostname": "R1",
    "loopback": "192.168.0.1"
  },
  {
    "hostname": "R2",
    "loopback": "192.168.0.2"
  },
  {
    "hostname": "R3",
    "loopback": "192.168.0.3"
  }
]
```

Complex Data: List of Dictionaries (Python)


```
[
  {
    "description": "DMVPN routers"
  },
  {
    "hostname": "R1",
    "loopback": "192.168.0.1"
  },
  {
    "hostname": "R2",
    "loopback": "192.168.0.2"
  },
  {
    "hostname": "R3",
    "loopback": "192.168.0.3"
  }
]
```

Dictionaries of Lists and Dictionaries

Complex Data: Lists and Dictionaries Within a Dictionary

```
---
hostname: R1
addresses:
  - 192.168.0.1
  - 192.168.1.1
loopback:
  ip: 192.168.0.1
  mask: 255.255.255.255
```

Starts in new line, indented

Value is a list

Value is a dictionary

A value of a dictionary key could be a

- A scalar value
- A list or dictionary
- A list of dictionaries …

Value starts in a new line, indented more than the key

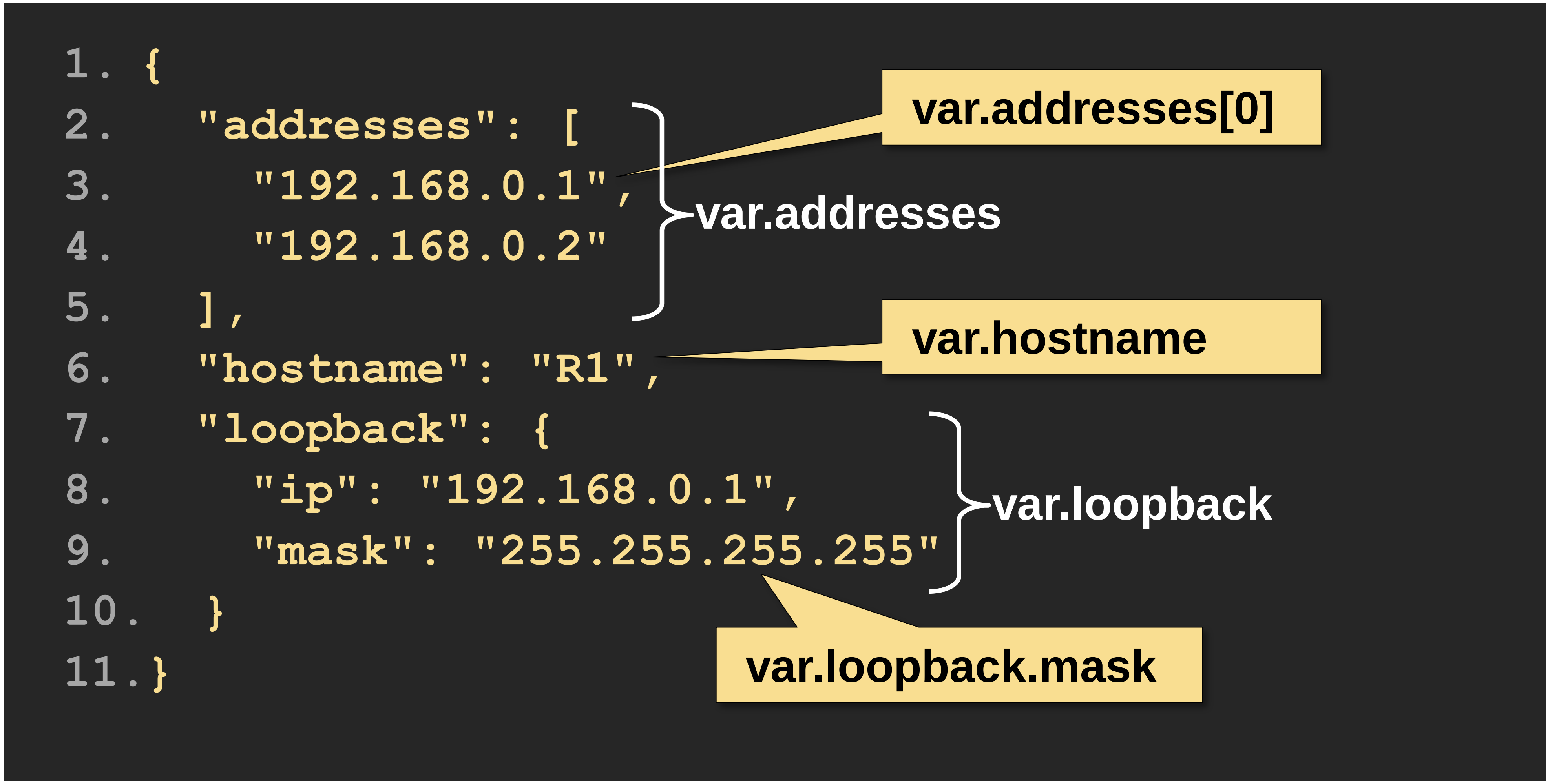
Lists and Dictionaries Within a Dictionary (JSON)

```
{
  "addresses": [
    "192.168.0.1",
    "192.168.0.2"
  ],
  "hostname": "R1",
  "loopback": {
    "ip": "192.168.0.1",
    "mask": "255.255.255.255"
  }
}
```

Value is a list

Value is a dictionary

Accessing Complex Lists and Dictionaries (Python)



```
{
  "addresses": [
    "192.168.0.1",
    "192.168.0.2"
  ],
  "hostname": "R1",
  "loopback": {
    "ip": "192.168.0.1",
    "mask": "255.255.255.255"
  }
}
```

YAML Shorthands

Inline Lists and Dictionaries

```
# Shorter version of
# router description
---
hostname: R1
addresses: [ 192.168.0.1, 192.168.0.2 ]
loopback: { ip: 192.168.0.1, mask: "/32" }
```

- List values listed in square brackets
- Dictionary values listed in curly brackets
- Recursion to any depth (hard to read though)
- Cannot be used at the document level

Hint: Inline data structures are in JSON format

Questions?

Send them to ip@ipSpace.net or @ioshints