YAML Ain't Markup Language

T. Pitner, L. Bártek, A. Rambousek. L Grolig
FI MU Brno 2020

YAML: What is it?

- “**YAML is a human friendly data serialization standard** for all programming languages.”

  https://yaml.org/ YAML home

  https://en.wikipedia.org/wiki/YAML Wikipedia YAML

- **Specification** (not a formal standard endorsed by ISO or W3C)

  https://yaml.org/spec/1.2/spec.html

YAML: Purpose, Goal

- "**YAML Ain't Markup Language**" (abbreviated YAML) is a data serialization language designed to be human-friendly and work well with modern programming languages for common everyday tasks."
- *Yet Another Markup Language*
- **`.yaml`** or **`.yml`** file extensions

YAML: Design principles

- It uses **Unicode** characters,
    - some provide structural information
    - rest containing the data itself, so it is a markup.
- YAML achieves a unique cleanness
- **Markup is minimal, content is maximal** => low overhead and natural look
- For example:
    - *spaces* (indentation) may be used for structure,
    - **key: value** pairs
    - dashes **-** for “bullet” lists

YAML: Design priorities

- *“YAML is easily **readable by humans**.*
- *YAML data is **portable** between programming languages.*
- *YAML matches the native data structures of agile languages.*
- *YAML has a consistent model to support generic tools.*
- *YAML supports **one-pass processing**.*
- *YAML is **expressive** and **extensible**.*
- *YAML is **easy** to implement and use.”*

YAML: Data types

- Three basic primitives
  - mappings (hashes/dictionaries),
  - sequences (arrays/lists) and
  - scalars (strings/numbers)

Everything else is a combination of above - and it is enough.

YAML: Usage

YAML was specifically created to work well for common use cases such as:

- **configuration** files,
- **log** files,
- interprocess **messaging**,
- cross-language **data sharing**,
- object **persistence**, and
- **debugging** of **complex data structures**.

YAML: Interfaces

- allows **incremental** (event-driven) interfaces
- **one-pass interfaces**
- thus enables processing of **large documents** (e.g. transaction logs) or
- **continuous streams** (e.g. feeds from a production machine)

YAML: Typing YAML documents

- Motivated by **Internet Mail (RFC0822)**
- C-style **escape sequences.** This enables ASCII encoding of non-printable or 8-bit (ISO 8859-1) characters such as **"\x3B"**. Non-printable 16-bit Unicode and 32-bit (ISO/IEC 10646) characters are supported with escape sequences such as **"\u003B"** and **"\U0000003B"**.
- A **single** line break is folded into a **single space,**
- while **empty lines** are interpreted as **line break** characters.
- This technique allows for paragraphs to be **word-wrapped** without affecting the canonical form of the scalar content.

YAML vs JSON

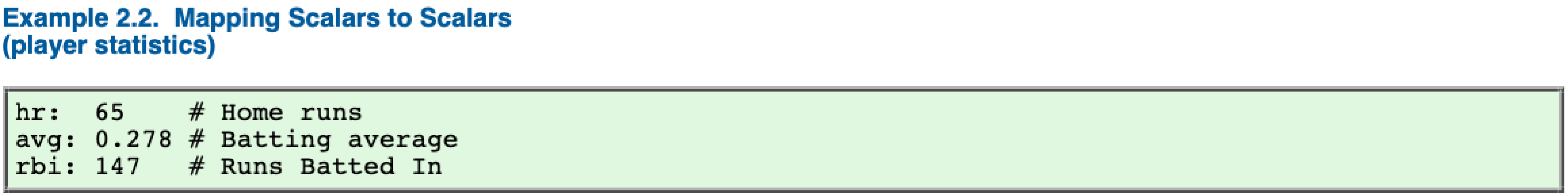
- YAML’s foremost design goals are **human readability**
- and support for serializing arbitrary **native data structures.**
- Extremely **readable** files, but is more **complex to generate and parse**
- YAML can therefore be viewed as a **natural superset of JSON**, offering improved human readability and a more complete information model.
- Every **JSON file** is also a **valid YAML file.**

YAML: Collections - Sequence

Example 2.1. Sequence of Scalars (ball players)

```
- Mark McGwire
- Sammy Sosa
- Ken Griffey
```

YAML: Map String -> Number



Example 2.2. Mapping Scalars to Scalars (player statistics)

```
hr:  65    # Home runs
avg: 0.278 # Batting average
rbi: 147   # Runs Batted In
```

… and comments after #

YAML: Map String -> Sequence

Example 2.3. Mapping Scalars to Sequences
(ball clubs in each league)

```
american:
  - Boston Red Sox
  - Detroit Tigers
  - New York Yankees
national:
  - New York Mets
  - Chicago Cubs
  - Atlanta Braves
```

YAML: Sequence of 2 maps

Example 2.4. Sequence of Mappings (players' statistics)

```
-
  name: Mark McGwire
  hr:   65
  avg:  0.278
-
  name: Sammy Sosa
  hr:   63
  avg:  0.288
```

YAML: Sequence of sequences

Example 2.5. Sequence of Sequences

```
- [name        , hr, avg  ]
- [Mark McGwire, 65, 0.278]
- [Sammy Sosa  , 63, 0.288]
```

YAML: Map of Maps

Example 2.6. Mapping of Mappings

```
Mark McGwire: {hr: 65, avg: 0.278}
Sammy Sosa: {
    hr: 63,
    avg: 0.288
  }
```

YAML: Basic syntactic rules

- Entire **Unicode character set**, except for some control characters, and may be encoded in **UTF-8, UTF-16 and UTF-32.**
- **Whitespace indentation** is used for denoting structure;
- tab characters are not allowed.
- **Comments** begin with the **number sign (#)**,
  can start anywhere on a line and continue until the end of the line.
- **List members** are denoted by a leading **hyphen (-)** with one member per line.
- A list can also be specified by enclosing text in **square brackets ([ ])** with each entry separated by commas.

YAML: Associative arrays

- associative array entry is **`key: value`**
- **`?key: value`** allows the key to contain leading dashes, square brackets, etc., without quotes.
- associative array can also be enclosed in JSON-style: **`{ key: value, key2: value2,... }`**, with

YAML: Scalar values

- Strings are *unquoted* or in double or single quotes **" '**
- Within double-quotes: C-style escape sequences \ may be used
- Octal escape is **\0**.
- Block scalars (longer texts) are delimited with indentation
- optional modifiers to preserve **|** or fold **>** newlines.

YAML: Big structures

- Multiple documents within a stream
- Start of document **---** and optional **…** end of document
- Nodes can be *named* **using** ampersand **&**
- and referenced with asterisk *
- label (type or tag) using **!!** followed by a string, which can be expanded into a URI.