*December 11, 2018 by Eric Goebelbecker* [*Leave a Comment*](https://rollout.io/blog/yaml-tutorial-everything-you-need-get-started/#respond)

YAML Ain’t Markup Language ([YAML](http://yaml.org/)) is a serialization language that has steadily increased in popularity over the last few years. It’s often used as a format for configuration files, but its object serialization abilities make it a viable replacement for languages like JSON. This YAML tutorial will demonstrate the language syntax with a guide and some simple coding examples in [Python](https://rollout.io/blog/python-feature-flag-guide/). YAML has broad language support and maps easily into native data structures. It’s also easy to for humans to read, which is why it’s a good choice for configuration.

The YAML acronym was shorthand for Yet Another Markup Language. But the maintainers renamed it to YAML Ain’t Markup Language to place more emphasis on its data-oriented features.

YAML Quick Start: A Simple File

Let’s take a look at a YAML file for a brief overview.

---

doe: "a deer, a female deer"

ray: "a drop of golden sun"

pi: 3.14159

xmas: true

french-hens: 3

calling-birds:

  - huey

  - dewey

  - louie

  - fred

xmas-fifth-day:

  calling-birds: four

  french-hens: 3

  golden-rings: 5

  partridges:

    count: 1

    location: "a pear tree"

  turtle-doves: two

The file starts with three dashes. These dashes indicate the start of a new YAML document. YAML supports multiple documents, and compliant parsers will recognize each set of dashes as the beginning of a new one.

Next, we see the construct that makes up most of a typical YAML document: a key-value pair. **Doe** is a key that points to a string value: **a deer, a female deer.**

YAML supports more than just string values. The file starts with six key-value pairs. They have four different data types. **Doe** and **ray** are strings. **Pi** is a floating-point number. **Xmas** is a boolean. **French-hens** is an integer. You can enclose strings in single or double-quotes or no quotes at all. YAML recognizes unquoted numerals as integers or floating point.

The seventh item is an array. **Calling-birds** has four elements, each denoted by an opening dash.

I indented the elements in **calling-birds** with two spaces. Indentation is how YAML denotes nesting. The number of spaces can vary from file to file, but tabs are not allowed. We’ll look at how indentation works below.

Finally, we see **xmas-fifth-day**, which has five more elements inside it, each of them indented. We can view **xmas-fifth-day** as a dictionary that contains two string, two integers, and another dictionary. YAML supports nesting of key-values, and mixing types.

Before we take a deeper dive, let’s look at how this document looks in JSON. I’ll throw it in this handy [JSON to YAML converter](https://www.json2yaml.com/).

{

 "doe": "a deer, a female deer",

 "ray": "a drop of golden sun",

 "pi": 3.14159,

 "xmas": true,

 "french-hens": 3,

 "calling-birds": [

    "huey",

    "dewey",

    "louie",

    "fred"

 ],

 "xmas-fifth-day": {

 "calling-birds": "four",

 "french-hens": 3,

 "golden-rings": 5,

 "partridges": {

   "count": 1,

   "location": "a pear tree"

 },

 "turtle-doves": "two"

 }

}

JSON and YAML have similar capabilities, and you can convert most documents between the formats.

Outline Indentation and Whitespace

Whitespace is part of YAML’s formatting. Unless otherwise indicated, newlines indicate the end of a field.

You structure a YAML document with indentation. The indentation level can be one or more spaces. The specification forbids tabs because tools treat them differently.

Consider this document. The items inside **stuff** are indented with two spaces.

foo: bar

    pleh: help

    stuff:

      foo: bar

      bar: foo

Let’s take a look at how a simple python script views this document. We’ll save it as a file named **foo.yaml**.

The PyYAML package will map a YAML file stream into a dictionary. We’ll iterate through the outermost set of keys and values and print the key and the string representation of each value. You can find a processor for your favorite platform here.

import yaml

if \_\_name\_\_ == '\_\_main\_\_':

   stream = open("foo.yaml", 'r')

   dictionary = yaml.load(stream)

   for key, value in dictionary.items():

       print (key + " : " + str(value))

The output is:

foo : bar

pleh : help

stuff : {'foo': 'bar', 'bar': 'foo'}

When we tell python to print a dictionary as a string, it uses the inline syntax we’ll see below. We can see from the output that our document is a python dictionary with two strings and another dictionary nested inside it.

testtttt

YAML’s simple nesting gives us the power to build sophisticated objects. But that’s only the beginning.

Comments

Comments begin with a pound sign. They can appear after a document value or take up an entire line.

\_\_\_

# This is a full line comment

foo: bar # this is a comment, too

Comments are for humans. YAML processors will discard them.

YAML Datatypes

Values in YAML’s key-value pairs are scalar. They act like the scalar types in languages like Perl, Javascript, and Python. It’s usually good enough to enclose strings in quotes, leave numbers unquoted, and let the parser figure it out.

But that’s only the tip of the iceberg. YAML is capable of a great deal more.

Key-Value Pairs and Dictionaries

The key-value is YAML’s basic building block. Every item in a YAML document is a member of at least one dictionary. The key is always a string. The value is a scalar so that it can be any datatype.

So, as we’ve already seen, the value can be a string, a number, or another dictionary.

Numeric types

YAML recognizes numeric types. We saw floating point and integers above. YAML supports several other numeric types.

An integer can be decimal, hexidecimal, or octal.

---

foo: 12345

bar: 0x12d4

plop: 023332

Let’s run our python script on this document.

foo : 12345

bar : 4820

plop : 9946

As you expect, **Ox** indicates a value is hex, and a leading zero denotes an octal value.

YAML supports both fixed and exponential floating point numbers.

---

foo: 1230.15

bar:  12.3015e+05

When we evaluate these entries we see:

foo : 1230.15

bar : 1230150.0

Finally, we can represent not-a-number (NAN) or infinity.

---

foo: .inf

bar: -.Inf

plop: .NAN

**Foo** is infinity. **Bar** is negative infinity, and **plop** is NAN.

Strings

YAML strings are Unicode. In most situations, you don’t have to specify them in quotes.

---

foo: this is a normal string

Our test program processes this as:

foo: this is a normal string

But if we want escape sequences handled, we need to use double quotes.

---

foo: "this is not a normal string\n"

bar: this is not a normal string\n

YAML processes the first value as ending with a carriage return and linefeed. Since the second value is not quoted, YAML treats the \n as two characters.

foo: this is not a normal string

bar : this is not a normal string\n

YAML will not escape strings with single quotes, but the single quotes do avoid having string contents interpreted as document formatting.

String values can span more than one line. With the fold (greater than) character, you can specify a string in a block.

bar: >

 this is not a normal string it

 spans more than

 one line

 see?

But it’s interpreted without the newlines.

bar : this is not a normal string it spans more than one line see?

The block (pipe) character has a similar function, but YAML interprets the field exactly as is.

bar: |

 this is not a normal string it

 spans more than

 one line

 see?

So, we see the newlines where they are in the document.

bar : this is not a normal string it

spans more than

one line

see?

Nulls

You enter nulls with a tilde or the unquoted null string literal.

---

foo: ~

bar: null

Our program prints:

foo : None

bar : None

Python’s representation for null is None.

Booleans

YAML indicates boolean values with the keywords True, On and Yes for true. False is indicated with False, Off, or No.

---

foo: True

bar: False

light: On

TV: Off

Arrays

You can specify arrays or lists on a single line.

---

items: [ 1, 2, 3, 4, 5 ]

names: [ "one", "two", "three", "four" ]

Or, you can put them on multiple lines.

---

items:

 - 1

 - 2

 - 3

 - 4

 - 5

names:

 - "one"

 - "two"

 - "three"

 - "four"

The multiple line format is useful for lists that contain complex objects instead of scalars.

\_\_\_

items:

 - things:

     thing1: huey

     things2: dewey

     thing3: louie

 - other things:

     key: value

An array can contain any valid YAML value. The values in a list do not have to be the same type.

Dictionaries

We covered dictionaries above, but there’s more to them.

Like arrays, you can put dictionaries inline. We saw this format above. It’s how python prints dictionaries.

---

foo: { thing1: huey, thing2: louie, thing3: dewey }

We’ve seen them span lines before.

---

foo: bar

bar: foo

And, of course, they can be nested and hold any value.

---

foo:

 bar:

   - bar

   - rab

   - plop

Advanced Options

Chomp Modifiers

Multiline values may end with whitespace, and depending on how you want the document to be processed you might not want to preserve it. YAML has the **strip** chomp and **preserve** chomp operators.

To save the last character, add a plus to the fold or block operators.

bar: >+

 this is not a normal string it

 spans more than

 one line

 see?

So, if the value ends with whitespace, like a newline, YAML will preserve it.

To strip the character, use the strip operator.

bar: |-

 this is not a normal string it

 spans more than

 one line

 see?

Multiple documents

A document starts with three dashes and ends with three periods. Some YAML processors require the document start operator. The end operator is usually optional. For example, Java’s Jackson will not process a YAML document without the start, but Python’s PyYAML will.

You’ll usually use the end document operator when a file contains multiple documents.

Let’s modify our python code.

import yaml

if \_\_name\_\_ == '\_\_main\_\_':

stream = open("foo.yaml", 'r')

dictionary = yaml.load\_all(stream)

for doc in dictionary:

print("New document:")

for key, value in doc.items():

print(key + " : " + str(value))

if type(value) is list:

print(str(len(value)))

PyYAML’s **load\_all** will process all of the documents in a stream.

Now, let’s process a compound document with it.

---

bar: foo

foo: bar

...

---

one: two

three: four

The script finds two YAML documents.

New document:

bar : foo

foo : bar

New document:

one : two

three : four

Conclusion

YAML is a powerful language that can be used for configuration files, messages between applications, and saving application state. We covered its most commonly used features, including how to use the built-in datatypes and structure complex documents. Some [platforms](http://yaml.org/) support YAML’s advanced features, including custom datatypes.