Learn X in Y minutes (/)

Where X=YAML

Get the code: learnyaml.yaml (/files/learnyaml.yaml)

YAML is a data serialisation language designed to be directly writable and readable by humans.

It's a strict superset of JSON, with the addition of syntactically significant newlines and indentation, like Python. Unlike Python, however, YAML doesn't allow literal tab characters for indentation.

```yaml
---  # document start

# Comments in YAML look like this.
# YAML supports single-line comments.

################
# SCALAR TYPES #
################

# Our root object (which continues for the entire document) will be a map,
# which is equivalent to a dictionary, hash or object in other languages.
key: value
another_key: Another value goes here.
a_number_value: 100
scientific_notation: 1e+12
hex_notation: 0x123  # evaluates to 291
octal_notation: 0123 # evaluates to 83

# The number 1 will be interpreted as a number, not a boolean.
# If you want it to be interpreted as a boolean, use true.
boolean: true
null_value: null
another_null_value: ~
key with spaces: value

# Yes and No (doesn't matter the case) will be evaluated to boolean
# true and false values respectively.
# To use the actual value use single or double quotes.
no: no            # evaluates to "no": false
yes: No           # evaluates to "yes": false
not_enclosed: yes # evaluates to "not_enclosed": true
enclosed: "yes"   # evaluates to "enclosed": yes

# Notice that strings don't need to be quoted. However, they can be.
however: 'A string, enclosed in quotes.'
'Keys can be quoted too.': "Useful if you want to put a ':' in your key."
single quotes: 'have ''one'' escape pattern'
double quotes: "have many: \", \0, \t, \u263A, \x0d\x0a == \r\n, and more."
# UTF-8/16/32 characters need to be encoded
Superscript two: \u00B2
```

```
# Special characters must be enclosed in single or double quotes
special_characters: "[ John ] & { Jane } - <Doe>"

# Multiple-line strings can be written either as a 'literal block' (using |),
# or a 'folded block' (using '>').
# Literal block turn every newline within the string into a literal newline (
# Folded block removes newlines within the string.
literal_block: |
  This entire block of text will be the value of the 'literal_block' key,
  with line breaks being preserved.

  The literal continues until de-dented, and the leading indentation is
  stripped.

      Any lines that are 'more-indented' keep the rest of their indentation -
      these lines will be indented by 4 spaces.
folded_style: >
  This entire block of text will be the value of 'folded_style', but this
  time, all newlines will be replaced with a single space.

  Blank lines, like above, are converted to a newline character.

      'More-indented' lines keep their newlines, too -
      this text will appear over two lines.

# |- and >- removes the trailing blank lines (also called literal/block "stri
literal_strip: |-
  This entire block of text will be the value of the 'literal_strip' key,
  with trailing blank line being stripped.
block_strip: >-
  This entire block of text will be the value of 'block_strip', but this
  time, all newlines will be replaced with a single space and
  trailing blank line being stripped.

# |+ and >+ keeps trailing blank lines (also called literal/block "keep")
literal_keep: |+
  This entire block of text will be the value of the 'literal_keep' key,
  with trailing blank line being kept.

block_keep: >+
  This entire block of text will be the value of 'block_keep', but this
  time, all newlines will be replaced with a single space and
```

```yaml
  trailing blank line being kept.

####################
# COLLECTION TYPES #
####################

# Nesting uses indentation. 2 space indent is preferred (but not required).
a_nested_map:
  key: value
  another_key: Another Value
  another_nested_map:
    hello: hello

# Maps don't have to have string keys.
0.25: a float key

# Keys can also be complex, like multi-line objects
# We use ? followed by a space to indicate the start of a complex key.
? |
  This is a key
  that has multiple lines
: and this is its value

# YAML also allows mapping between sequences with the complex key syntax
# Some language parsers might complain
# An example
? - Manchester United
  - Real Madrid
: [ 2001-01-01, 2002-02-02 ]

# Sequences (equivalent to lists or arrays) look like this
# (note that the '-' counts as indentation):
a_sequence:
  - Item 1
  - Item 2
  - 0.5  # sequences can contain disparate types.
  - Item 4
  - key: value
    another_key: another_value
  - - This is a sequence
    - inside another sequence
  - - - Nested sequence indicators
```

```
      - can be collapsed

# Since YAML is a superset of JSON, you can also write JSON-style maps and
# sequences:
json_map: { "key": "value" }
json_seq: [ 3, 2, 1, "takeoff" ]
and quotes are optional: { key: [ 3, 2, 1, takeoff ] }

#######################
# EXTRA YAML FEATURES #
#######################

# YAML also has a handy feature called 'anchors', which let you easily duplic
# content across your document.
# Anchors identified by & character which define the value.
# Aliases identified by * character which acts as "see above" command.
# Both of these keys will have the same value:
anchored_content: &anchor_name This string will appear as the value of two ke
other_anchor: *anchor_name

# Anchors can be used to duplicate/inherit properties
base: &base
  name: Everyone has same name

# The expression << is called 'Merge Key Language-Independent Type'. It is us
# indicate that all the keys of one or more specified maps should be insertec
# into the current map.
# NOTE: If key already exists alias will not be merged
foo:
  <<: *base # doesn't merge the anchor
  age: 10
  name: John
bar:
  <<: *base # base anchor will be merged
  age: 20

# foo name won't be changed and it will be: John. On the other hand, bar's na

# YAML also has tags, which you can use to explicitly declare types.
# Syntax: !![typeName] [value]
explicit_boolean: !!bool true
explicit_integer: !!int 42
```

```yaml
explicit_float: !!float -42.24
explicit_string: !!str 0.5
explicit_datetime: !!timestamp 2022-11-17 12:34:56.78 +9
explicit_null: !!null null

# Some parsers implement language specific tags, like this one for Python's
# complex number type.
python_complex_number: !!python/complex 1+2j

# We can also use yaml complex keys with language specific tags
? !!python/tuple [ 5, 7 ]
: Fifty Seven
# Would be {(5, 7): 'Fifty Seven'} in Python

####################
# EXTRA YAML TYPES #
####################

# Strings and numbers aren't the only scalars that YAML can understand.
# ISO-formatted date and datetime literals are also parsed.
datetime_canonical: 2001-12-15T02:59:43.1Z
datetime_space_separated_with_time_zone: 2001-12-14 21:59:43.10 -5
date_implicit: 2002-12-14
date_explicit: !!timestamp 2002-12-14

# The !!binary tag indicates that a string is actually a base64-encoded
# representation of a binary blob.
gif_file: !!binary |
  R0lGODlhDAAMAIQAAP//9/X17unp5WZmZgAAAOfn515eXvPz7Y6OjuDg4J+fn5
  OTk6enp56enmlpaWNjY6Ojo4SEhP/++f/++f/++f/++f/++f/++f/++f/++f/+
  +f/++f/++f/++f/++f/++SH+Dk1hZGUgd2l0aCBHSU1QACwAAAAADAAMAAAFLC
  AgjoEwnuNAFOhpEMTRiggcz4BNJHrv/zCFcLiwMWYNG84BwwEeECcgggoBADs=

# YAML also has a set type, which looks like this:
set:
  ? item1
  ? item2
  ? item3
or: { item1, item2, item3 }

# Sets are just maps with null values; the above is equivalent to:
set2:
  item1: null
  item2: null
  item3: null

...  # document end
```

More Resources

- YAML official website (https://yaml.org/)
- Online YAML Validator (http://www.yamllint.com/)
- JSON ⇆ YAML (https://www.json2yaml.com/)
