Learn X in Y minutes (/)

Where X=toml

Get the code: learntoml.toml (/docs/files/learntoml.toml)

TOML stands for Tom's Obvious, Minimal Language. It is a data serialisation language designed to be a minimal configuration file format that's easy to read due to obvious semantics.

It is an alternative to YAML and JSON. It aims to be more human friendly than JSON and simpler that YAML. TOML is designed to map unambiguously to a hash table. TOML should be easy to parse into data structures in a wide variety of languages.

Be warned, TOML's spec is still changing a lot. Until it's marked as 1.0, you should assume that it is unstable and act accordingly. This document follows TOML v0.4.0.

```
# Comments in TOML look like this.

################
# 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.

# The key, equals sign, and value must be on the same line
# (though some values can be broken over multiple lines).
key = "value"
string = "hello"
number = 42
float = 3.14
boolean = true
dateTime = 1979-05-27T07:32:00-08:00
scientificNotation = 1e+12
"key can be quoted" = true # Both " and ' are fine
"key may contain" = "letters, numbers, underscores, and
dashes"

# A bare key must be non-empty, but an empty quoted key is
allowed
"" = "blank"     # VALID but discouraged
'' = 'blank'     # VALID but discouraged

##########
# String #
##########

# All strings must contain only valid UTF-8 characters.
# We can escape characters and some of them have a compact
escape sequence.
# For example, \t add a tabulation. Refers to the spec to
get all of them.
basicString = "are surrounded by quotation marks. \"I'm
quotable\". Name\tJos"

multiLineString = """
are surrounded by three quotation marks
on each side and allow newlines."""
```

```
literalString = 'are surrounded by single quotes. Escaping
are not allowed.'

multiLineLiteralString = '''
are surrounded by three single quotes on each side
and allow newlines. Still no escaping.
The first newline is trimmed in raw strings.
   All other whitespace
   is preserved. #! are preserved?
'''

# For binary data it is recommended that you use Base64,
another ASCII or UTF8
# encoding. The handling of that encoding will be
application specific.

###########
# Integer #
###########

## Integers can start with a +, a - or nothing.
## Leading zeros are not allowed. Hex, octal, and binary
forms are not allowed.
## Values that cannot be expressed as a series of digits
are not allowed.
int1 = +42
int2 = 0
int3 = -21
integerRange = 64

## You can use underscores to enhance readability. Each
## underscore must be surrounded by at least one digit.
int4 = 5_349_221
int5 = 1_2_3_4_5     # VALID but discouraged

#########
# Float #
#########

# Floats are an integer followed by a fractional and/or an
exponent part.
flt1 = 3.1415
flt2 = -5e6
flt3 = 6.626E-34
```

```
###########
# Boolean #
###########

bool1 = true
bool2 = false
boolMustBeLowercase = true

############
# Datetime #
############

date1 = 1979-05-27T07:32:00Z # UTC time, following RFC
3339/ISO 8601 spec
date2 = 1979-05-26T15:32:00+08:00 # with RFC 3339/ISO 8601
offset
date3 = 1979-05-27T07:32:00 # without offset
date4 = 1979-05-27 # without offset or time

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

#########
# Array #
#########

array1 = [ 1, 2, 3 ]
array2 = [ "Commas", "are", "delimiters" ]
array3 = [ "Don't mix", "different", "types" ]
array4 = [ [ 1.2, 2.4 ], ["all", 'strings', """are the
same""", '''type'''] ]
array5 = [
  "Whitespace", "is", "ignored"
]

#########
# Table #
#########

# Tables (or hash tables or dictionaries) are collections
of key/value
# pairs. They appear in square brackets on a line by
themselves.
# Empty tables are allowed and simply have no key/value
```

```
pairs within them.
[table]

# Under that, and until the next table or EOF are the
key/values of that table.
# Key/value pairs within tables are not guaranteed to be in
any specific order.
[table-1]
key1 = "some string"
key2 = 123

[table-2]
key1 = "another string"
key2 = 456

# Dots are prohibited in bare keys because dots are used to
signify nested tables.
# Naming rules for each dot separated part are the same as
for keys.
[dog."tater.man"]
type = "pug"

# In JSON land, that would give you the following
structure:
# { "dog": { "tater.man": { "type": "pug" } } }

# Whitespace around dot-separated parts is ignored,
however, best practice is to
# not use any extraneous whitespace.
[a.b.c]            # this is best practice
[ d.e.f ]          # same as [d.e.f]
[ j . "ʞ" . 'l' ]  # same as [j."ʞ".'l']

# You don't need to specify all the super-tables if you
don't want to. TOML knows
# how to do it for you.
# [x] you
# [x.y] don't
# [x.y.z] need these
[x.y.z.w] # for this to work

# As long as a super-table hasn't been directly defined and
hasn't defined a
# specific key, you may still write to it.
[a.b]
```

```
c = 1

[a]
d = 2

# Will generate the following in JSON:
# { "a": {"b": {"c": 1}, "d": 2 } }

# You cannot define any key or table more than once. Doing
so is invalid.

# DO NOT DO THIS
[a]
b = 1

[a]
c = 2

# DO NOT DO THIS EITHER
[a]
b = 1

[a.b]
c = 2

# All table names must be non-empty.
[]     # INVALID
[a.]   # INVALID
[a..b] # INVALID
[.b]   # INVALID
[.]    # INVALID

################
# Inline table #
################

inlineTables = { areEnclosedWith = "{ and }", mustBeInline
= true }
point = { x = 1, y = 2 }

####################
# Array of Tables #
####################

# An array of tables can be expressed by using a table name
```

```
in double brackets.
# Each table with the same double bracketed name will be an
item in the array.
# The tables are inserted in the order encountered.

[[products]]
name = "array of table"
sku = 738594937
emptyTableAreAllowed = true

[[products]]

[[products]]
name = "Nail"
sku = 284758393
color = "gray"
```

The equivalent in JSON would be:

```
{
  "products": [
    {
      "name": "array of table",
      "sku": 7385594937,
      "emptyTableAreAllowed": true
    },
    {},
    {
      "name": "Nail",
      "sku": 284758393,
      "color": "gray"
    }
  ]
}
```

```
# You can create nested arrays of tables as well. Each
double-bracketed
# sub-table will belong to the nearest table element above
it.

[[fruit]]
  name = "apple" # I am a property in fruit table/map

  [fruit.geometry]
    shape = "round"
    note = "I am a property in geometry table/map"

  [[fruit.color]]
    name = "red"
    note = "I am an array item in apple fruit's table/map"

  [[fruit.color]]
    name = "green"
    note = "I am in the same array as red"

[[fruit]]
  name = "banana"

  [[fruit.color]]
    name = "yellow"
    note = "I am an array item in banana fruit's table/map"
```

The equivalent in JSON would be:

```
{
  "fruit": [
    {
      "name": "apple",
      "geometry": { "shape": "round", "note": "..."},
      "color": [
        { "name": "red", "note": "..." },
        { "name": "green", "note": "..." }
      ]
    },
    {
      "name": "banana",
      "color": [
        { "name": "yellow", "note": "..." }
      ]
    }
  ]
}
```

More Resources

- TOML official repository (https://github.com/toml-lang/toml)

Got a suggestion? A correction, perhaps? Open an Issue (https://github.com/adambard/learnxinyminutes-docs/issues/new) on the Github Repo, or make a pull request (https://github.com/adambard/learnxinyminutes-docs/edit/master/toml.html.markdown) yourself!

Originally contributed by Alois de Gouvello, and updated by 6 contributor(s) (https://github.com/adambard/learnxinyminutes-docs/blame/master/toml.html.markdown).