Introduction to TOML

Tom's Obvious, Minimal Language

What is TOML?

Configuration file format that's easy to read and write

- **Application configuration** (Rust Cargo, Python Poetry)
- **Project metadata** (pyproject.toml, Cargo.toml)
- **Settings files** for modern development tools

TOML aims to be a **minimal configuration file format** that's easy to read due to obvious semantics.

Basic Syntax Rules

- TOML has `key = value` pairs (note the `=` not `:`)
- Comments start with #
- Section headers use `[section_name]`

```toml
# This is a comment
name = "John Smith"              # String
age = 25                         # Number
is_student = true                # Boolean
graduation_date = 1979-05-27 # Date
```

TOML vs YAML: Basic Values

TOML:

```toml
name = "John Smith"
age = 25
is_student = true
graduation = null  # not supported in TOML
```

YAML:

```yaml
name: "John Smith"
age: 25
is_student: true
graduation: null
```

Key Difference: TOML uses `=`, YAML uses `:`

Arrays in TOML vs YAML

TOML:

```
courses = ["Computer Science", "Mathematics", "Physics"]
courses = [
    "Computer Science",
    "Mathematics",
    "Physics"
]
```

YAML:

```
courses:
  - "Computer Science"
  - "Mathematics"
  - "Physics"
```

Key Difference: TOML uses brackets `[]`, YAML uses indentation with `-`

Tables (Objects) in TOML vs YAML

TOML:

```toml
[address]
street = "123 Main St"
city = "Boston"
zip = 02101
```

YAML:

```yaml
address:
  street: "123 Main St"
  city: "Boston"
  zip: 02101
```

Key Difference: TOML uses `[section]` headers, YAML uses indentation

Inline Tables in TOML

TOML also supports inline tables:

```toml
address = { street = "123 Main St", city = "Boston", zip = 02101 }
```

Equivalent YAML:

```yaml
address:
  street: "123 Main St"
  city: "Boston"
  zip: 02101
```

Inline tables are helpful for **simple, related data** that fits on one line.

Multi-line Strings

TOML:

```toml
description = """
This is a multi-line
description that preserves
line breaks.
"""

summary = """
This is a long text \
that continues on \
the next line."""
```

YAML:

```yaml
description: |
  This is a multi-line
  description that preserves
  line breaks.

summary: >
  This is a long text
  that will be folded
  into a single line.
```

Array of Tables

TOML's unique feature:

```toml
[[products]]
name = "Hammer"
sku = 738594937

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

Equivalent YAML:

```yaml
products:
  - name: "Hammer"
    sku: 738594937
  - name: "Nail"
    sku: 284758393
```

Key Advantage: TOML's [[table]] syntax is more explicit for

Nested Tables

TOML:

```toml
[database]
server = "192.168.1.1"
ports = [8001, 8001, 8002]

[database.connection]
max_retry = 3
timeout = 5000
```

YAML:

```yaml
database:
  server: "192.168.1.1"
  ports: [8001, 8001, 8002]
  connection:
    max_retry: 3
    timeout: 5000
```

Real-World Example: Rust Cargo.toml

```toml
[package]
name = "my_rust_app"
version = "0.1.0"
edition = "2021"

[dependencies]
serde = { version = "1.0", features = ["derive"] }
tokio = { version = "1.0", features = ["full"] }

[[bin]]
name = "server"
path = "src/server.rs"

[[bin]]
name = "client"
path = "src/client.rs"
```

Equivalent in YAML

```yaml
package:
  name: "my_rust_app"
  version: "0.1.0"
  edition: "2021"

dependencies:
  serde:
    version: "1.0"
    features: ["derive"]
  tokio:
    version: "1.0"
    features: ["full"]

bin:
  - name: "server"
    path: "src/server.rs"
  - name: "client"
    path: "src/client.rs"
```

Python Poetry Example

pyproject.toml:

```toml
[tool.poetry]
name = "my-python-app"
version = "0.1.0"
description = "A sample Python project"

[tool.poetry.dependencies]
python = "^3.8"
requests = "^2.28.0"
fastapi = "^0.68.0"

[tool.poetry.dev-dependencies]
pytest = "^6.0.0"
black = "^21.0.0"
```

Key Differences Summary

Feature	TOML	YAML
Syntax	`key = value`	`key: value`
Arrays	`[item1, item2]`	`- item1` `- item2`
Objects	`[section]`	indentation
Comments	`# comment`	`# comment`
Multi-line	`"""text"""`	`|` or `>`
Readability	Excellent	Excellent
Learning Curve	Easy	Moderate

When to Use TOML vs YAML

Use TOML for:

- **Application configuration** (especially Rust, Python)
- **Project metadata** files
- **Settings** that won't be deeply nested
- When you want **explicit structure**

Use YAML for:

- **DevOps** (Docker, Kubernetes, CI/CD)
- **Documentation** frontmatter
- **Complex nested data** structures
- When you need **advanced features** (anchors, references)

Real-World Usage

TOML Adoption:

- **Rust ecosystem** - Cargo.toml (universal)
- **Python packaging** - pyproject.toml (PEP 518)
- **Hugo static sites** - config.toml
- **Prettier code formatter** - .prettierrc.toml

YAML Still Dominates:

- **DevOps tooling** (Kubernetes, Docker)
- **CI/CD pipelines** (GitHub Actions, GitLab)
- **Documentation** (Jekyll, Hugo frontmatter)

TOML Data Types

TOML supports more specific types than YAML:

```toml
# Strings
basic_string = "I'm a string"
literal_string = 'C:\Users\nodejs\templates'

# Numbers
integer = 123
float = 123.45
hex = 0xDEADBEEF
octal = 0o755
binary = 0b11010110

# Dates (ISO 8601)
date = 1979-05-27
datetime = 1979-05-27T07:32:00Z
local_datetime = 1979-05-27T07:32:00
local_time = 07:32:00
```

Conclusion

Use Case	Recommended Format	Why
Rust Projects	TOML	Standard ecosystem choice
Python Packaging	TOML	PEP 518 standard
DevOps	YAML	Industry standard
Simple Config	TOML	Less indentation errors
Complex Config	YAML	More flexible structure
CI/CD Pipelines	YAML	Tool ecosystem

Choose the Right Tool

TOML for simple, explicit configuration
YAML for complex, flexible data structures

Both are human-readable and better than JSON for configuration.