Formatting & Branding

Session 5

Mickaël CANOUIL, *Ph.D.*

pro@mickael.canouil.dev

Independent Contractor

Saturday, the 7th of February, 2026

 |

- Session 5: Formatting & Branding
- Session Objectives
- Learning Objectives

Session Overview

 |

Session 5: Formatting & Branding

- **Bootstrap Theming & Customisation**
  - Bootstrap 5 integration and built-in theme selection.
  - Progressive customisation from YAML options to light CSS.
  - Theme layering and CSS custom properties.
- **Pandoc Templating & Extensions**
  - Understanding Pandoc's templating syntax fundamentals.
  - Variables, conditionals, and loops for dynamic content.
  - Developing brand extensions for organisational reuse.
- **Unified Branding with Brand.yml**
  - Brand.yml systems for cross-format consistency.
  - Colour palettes, typography, and logo management.
  - Integration with Bootstrap and format-specific options.
- **Typst Customisation**
  - Typst templates and partials for PDF branding.
  - Template structure and brand.yml integration.
  - Creating custom Typst formats with consistent visual identity.

Session Objectives

This session explores Quarto's advanced theming and branding capabilities, covering Bootstrap 5 customisation, `brand.yml` systems for unified branding, Pandoc templating syntax, and Typst customisation for professional PDF output.

 |

Learning Objectives

 |

Learning Objectives

By the end of this session, participants will be able to:

 |

Learning Objectives

By the end of this session, participants will be able to:

- Implement professional themes with Bootstrap 5 customisation.

 |

Learning Objectives

By the end of this session, participants will be able to:

- Implement professional themes with Bootstrap 5 customisation.
- Apply progressive customisation from YAML to light CSS.

 |

Learning Objectives

By the end of this session, participants will be able to:

- Implement professional themes with Bootstrap 5 customisation.
- Apply progressive customisation from YAML to light CSS.
- Create unified branding across multiple output formats using `brand.yml`.

 |

Learning Objectives

By the end of this session, participants will be able to:

- Implement professional themes with Bootstrap 5 customisation.
- Apply progressive customisation from YAML to light CSS.
- Create unified branding across multiple output formats using `brand.yml`.
- Develop brand extensions for organisational reuse.

 |

Learning Objectives

By the end of this session, participants will be able to:

- Implement professional themes with Bootstrap 5 customisation.
- Apply progressive customisation from YAML to light CSS.
- Create unified branding across multiple output formats using `brand.yml`.
- Develop brand extensions for organisational reuse.
- Understand Pandoc's templating syntax for dynamic content generation.

 |

Learning Objectives

By the end of this session, participants will be able to:

- Implement professional themes with Bootstrap 5 customisation.
- Apply progressive customisation from YAML to light CSS.
- Create unified branding across multiple output formats using `brand.yml`.
- Develop brand extensions for organisational reuse.
- Understand Pandoc's templating syntax for dynamic content generation.
- Customise Typst templates and partials for branded PDF output.

 |

- Bootstrap in Quarto
- Theme Selection
- Basic Customisation
- Progressive Enhancement

Theming (Bootstrap & Customisation)

 |

Bootstrap in Quarto

 |

Bootstrap in Quarto

Quarto HTML format uses Bootstrap 5, providing:

 |

Bootstrap in Quarto

Quarto HTML format uses Bootstrap 5, providing:

- **Several built-in themes** from Bootswatch.

Bootstrap in Quarto

Quarto HTML format uses Bootstrap 5, providing:

- **Several built-in themes** from Bootswatch.
- **Responsive grid system**.

 |

Bootstrap in Quarto

Quarto HTML format uses Bootstrap 5, providing:

- **Several built-in themes** from Bootswatch.
- **Responsive grid system**.
- **Component library** (*e.g.*, buttons, cards, navigation).

 |

Bootstrap in Quarto

Quarto HTML format uses Bootstrap 5, providing:

- **Several built-in themes** from Bootswatch.
- **Responsive grid system**.
- **Component library** (*e.g.*, buttons, cards, navigation).
- **Customisation system** via CSS variables.

Theme Selection

 |

Theme Selection

Available themes: default, cerulean, cosmo, cyborg, darkly, flatly, journal, litera, lumen, lux, materia, minty, morph, pulse, quartz, sandstone, simplex, sketchy, slate, solar, spacelab, superhero, united, vapor, yeti, zephyr.

Theme Selection

Available themes: default, cerulean, cosmo, cyborg, darkly, flatly, journal, litera, lumen, lux, materia, minty, morph, pulse, quartz, sandstone, simplex, sketchy, slate, solar, spacelab, superhero, united, vapor, yeti, zephyr.

See HTML Theming for more.

Theme Selection

- Single theme.

YAML

```yaml
format:
  html:
    theme: cosmo
```

 |

Theme Selection

- Light/dark theme pair.

YAML

```yaml
format:
  html:
    theme:
      light: cosmo
      dark: darkly
```

Theme Selection

- Theme + customisation.

YAML

```yaml
format:
  html:
    theme: [cosmo, custom.scss]
```

 |

Theme Selection

Can also be written as:

YAML

```yaml
format:
  html:
    theme:
      - cosmo
      - custom.scss
```

 |

Basic Customisation

YAML

```yaml
format:
  html:
    theme: cosmo

    # Typography
    fontsize: 1.1em
    linestretch: 1.6
    mainfont: "Inter"
```

 |

Basic Customisation

YAML

```yaml
format:
  html:
    theme: cosmo

    # Typography
    fontsize: 1.1em
    linestretch: 1.6
    mainfont: "Inter"

    # Colours
    linkcolor: "#2E86AB"
    backgroundcolor: "#FFFFFF"
```

 |

Basic Customisation

YAML

```yaml
format:
  html:
    theme: cosmo

    # Typography
    fontsize: 1.1em
    linestretch: 1.6
    mainfont: "Inter"

    # Colours
    linkcolor: "#2E86AB"
    backgroundcolor: "#FFFFFF"

    # Layout
    max-width: 1200px
    margin-left: 2rem
```

 |

Basic Customisation

YAML

```yaml
format:
  html:
    theme: cosmo

    # Typography
    fontsize: 1.1em
    linestretch: 1.6
    mainfont: "Inter"

    # Colours
    linkcolor: "#2E86AB"
    backgroundcolor: "#FFFFFF"

    # Layout
    max-width: 1200px
    margin-left: 2rem
```

See HTML Theming - Basic Options for more.

Progressive Enhancement

Move to SCSS for more control:

YAML

```yaml
format:
  html:
    theme:
      - cosmo
      - styles.scss
```

Custom SCSS file follows a specific structure using comments:

styles.scss

```scss
/*-- scss:defaults --*/
$h2-font-size:          1.6rem !default;
$headings-font-weight:  500 !default;

/*-- scss:rules --*/
h1, h2, h3, h4, h5, h6 {
  text-shadow: -1px -1px 0 rgba(0, 0, 0, .3);
}
```

See More About Quarto Themes for more.

`brand.yml` Foundation

- Understanding Brand.yml
- Basic brand.yml Structure
- Colour Palette System
- Typography Configuration
- Logo Management
- Brand Integration

 |

Understanding Brand.yml

 |

Understanding Brand.yml

`brand.yml` is a single YAML file that codifies your organisation's brand guidelines into a format that can be used by Quarto (*but not limited to*) to create branded outputs across multiple formats.

Understanding Brand.yml

`brand.yml` is a single YAML file that codifies your organisation's brand guidelines into a format that can be used by Quarto (*but not limited to*) to create branded outputs across multiple formats.

Key benefits:

 |

Understanding Brand.yml

`brand.yml` is a single YAML file that codifies your organisation's brand guidelines into a format that can be used by Quarto (*but not limited to*) to create branded outputs across multiple formats.

Key benefits:

- **Cross-format consistency** - HTML, PDF, presentations share the same design.

 |

Understanding Brand.yml

`brand.yml` is a single YAML file that codifies your organisation's brand guidelines into a format that can be used by Quarto (*but not limited to*) to create branded outputs across multiple formats.

Key benefits:

- **Cross-format consistency** - HTML, PDF, presentations share the same design.
- **Organisational standards** - Teams work with unified branding.

 |

Understanding Brand.yml

`brand.yml` is a single YAML file that codifies your organisation's brand guidelines into a format that can be used by Quarto (*but not limited to*) to create branded outputs across multiple formats.

Key benefits:

- **Cross-format consistency** - HTML, PDF, presentations share the same design.
- **Organisational standards** - Teams work with unified branding.
- **Easy maintenance** - Update once, apply everywhere.

 |

Understanding Brand.yml

`brand.yml` is a single YAML file that codifies your organisation's brand guidelines into a format that can be used by Quarto (*but not limited to*) to create branded outputs across multiple formats.

Key benefits:

- **Cross-format consistency** - HTML, PDF, presentations share the same design.
- **Organisational standards** - Teams work with unified branding.
- **Easy maintenance** - Update once, apply everywhere.
- **Simple syntax** - No complex CSS knowledge required.

Basic `brand.yml` Structure

Create a `_brand.yml` file in your project root:

_brand.yml

```yaml
meta:
  name: "Your Organisation"
  link: "https://yourorg.com"

color:
  palette:
    primary: "#2E86AB"
    secondary: "#A23B72"
    light: "#F8F9FA"
    dark: "#343A40"
  primary: primary
  foreground: dark
  background: light

logo:
  small: "assets/logo-small.png"
  medium: "assets/logo-medium.png"
  large: "assets/logo-large.png"

typography:
  fonts:
    - family: "Inter"
```

Basic `brand.yml` Structure

Create a `_brand.yml` file in your project root:

_brand.yml

```yaml
meta:
  name: "Your Organisation"
  link: "https://yourorg.com"

color:
  palette:
    primary: "#2E86AB"
    secondary: "#A23B72"
    light: "#F8F9FA"
    dark: "#343A40"
  primary: primary
  foreground: dark
  background: light

logo:
  small: "assets/logo-small.png"
  medium: "assets/logo-medium.png"
  large: "assets/logo-large.png"

typography:
  fonts:
    - family: "Inter"
```

1 Basic brand identity.

2

3

4

Basic `brand.yml` Structure

Create a `_brand.yml` file in your project root:

_brand.yml

```yaml
meta:
  name: "Your Organisation"
  link: "https://yourorg.com"

color:
  palette:
    primary: "#2E86AB"
    secondary: "#A23B72"
    light: "#F8F9FA"
    dark: "#343A40"
  primary: primary
  foreground: dark
  background: light

logo:
  small: "assets/logo-small.png"
  medium: "assets/logo-medium.png"
  large: "assets/logo-large.png"

typography:
  fonts:
    - family: "Inter"
```

(1)

(2) Colour palette system.

(3)

(4)

Basic `brand.yml` Structure

Create a `_brand.yml` file in your project root:

_brand.yml

```yaml
color:
  palette:
    primary: "#2E86AB"
    secondary: "#A23B72"
    light: "#F8F9FA"
    dark: "#343A40"
  primary: primary
  foreground: dark
  background: light

logo:
  small: "assets/logo-small.png"
  medium: "assets/logo-medium.png"
  large: "assets/logo-large.png"

typography:
  fonts:
    - family: "Inter"
      source: "google"
  base: "Inter"
  headings: "Inter"
```

(2)

(3) Logo management.

(4)

Basic `brand.yml` Structure

Create a `_brand.yml` file in your project root:

_brand.yml

```yaml
color:
  palette:
    primary: "#2E86AB"
    secondary: "#A23B72"
    light: "#F8F9FA"
    dark: "#343A40"
  primary: primary
  foreground: dark
  background: light

logo:
  small: "assets/logo-small.png"
  medium: "assets/logo-medium.png"
  large: "assets/logo-large.png"

typography:
  fonts:
    - family: "Inter"
      source: "google"
  base: "Inter"
  headings: "Inter"
```

(2)

(3)

(4) Typography system.

Basic `brand.yml` Structure

Create a `_brand.yml` file in your project root:

_brand.yml

```yaml
meta:
  name: "Your Organisation"
  link: "https://yourorg.com"

color:
  palette:
    primary: "#2E86AB"
    secondary: "#A23B72"
    light: "#F8F9FA"
    dark: "#343A40"
  primary: primary
  foreground: dark
  background: light

logo:
  small: "assets/logo-small.png"
  medium: "assets/logo-medium.png"
  large: "assets/logo-large.png"

typography:
  fonts:
    - family: "Inter"
```

Colour Palette System

 |

Colour Palette System

The palette defines named colours that can be referenced throughout your brand:

_brand.yml

```yaml
color:
  palette:
    ocean-blue: "#2E86AB"
    coral-pink: "#A23B72"
    sunshine: "#F4A261"
    forest: "#2A9D8F"
    charcoal: "#264653"
    pearl: "#F8F9FA"
  # Semantic assignments
  primary: ocean-blue
  secondary: coral-pink
  success: forest
  warning: sunshine
  foreground: charcoal
  background: pearl
```

Colour Palette System

The palette defines named colours that can be referenced throughout your brand:

_brand.yml

```yaml
color:
  palette:
    ocean-blue: "#2E86AB"
    coral-pink: "#A23B72"
    sunshine: "#F4A261"
    forest: "#2A9D8F"
    charcoal: "#264653"
    pearl: "#F8F9FA"
  # Semantic assignments
  primary: ocean-blue
  secondary: coral-pink
  success: forest
  warning: sunshine
  foreground: charcoal
  background: pearl
```

Cross-format usage:

- **HTML**: Available as `$brand-primary` (SCSS), `--brand-primary` (CSS), etc.
- **Typst**: Available as `brand-color.primary`, etc.

Colour Palette System

The palette defines named colours that can be referenced throughout your brand:

_brand.yml

```yaml
color:
  palette:
    ocean-blue: "#2E86AB"
    coral-pink: "#A23B72"
    sunshine: "#F4A261"
    forest: "#2A9D8F"
    charcoal: "#264653"
    pearl: "#F8F9FA"
  # Semantic assignments
  primary: ocean-blue
  secondary: coral-pink
  success: forest
  warning: sunshine
  foreground: charcoal
  background: pearl
```

1 Define a palette of named co

Cross-format usage:

- **HTML**: Available as `$brand-primary` (SCSS), `--brand-primary` (CSS), etc.
- **Typst**: Available as `brand-color.primary`, etc.

Colour Palette System

The palette defines named colours that can be referenced throughout your brand:

_brand.yml

```yaml
color:
  palette:
    ocean-blue: "#2E86AB"
    coral-pink: "#A23B72"
    sunshine: "#F4A261"
    forest: "#2A9D8F"
    charcoal: "#264653"
    pearl: "#F8F9FA"
  # Semantic assignments
  primary: ocean-blue
  secondary: coral-pink
  success: forest
  warning: sunshine
  foreground: charcoal
  background: pearl
```

Cross-format usage:

- **HTML**: Available as `$brand-primary` (SCSS), `--brand-primary` (CSS), etc.
- **Typst**: Available as `brand-color.primary`, etc.

Typography Configuration

Define your organisation's font system:

YAML

```yaml
typography:
  fonts:
    # Google Fonts (automatically loaded)
    - family: "Inter"
      source: "google"
      weight: [300, 400, 500, 600, 700]

    # System fonts (fallback)
    - family: "system-ui"
      source: "system"

  # Font assignments
  base: "Inter"             # Body text
  headings: "Inter"         # All headings
  monospace: "SF Mono"      # Code blocks
```

See Typst Brand YAML for troubleshooting fonts in Typst.

Logo Management

 |

Logo Management

Provide logos at different sizes for various contexts:

YAML

```yaml
logo:
  # Favicon and small contexts
  small: "brand/favicon.png"

  # Navigation bars, headers
  medium: "brand/logo-nav.png"

  # Hero sections, print materials
  large: "brand/logo-full.png"
```

Logo Management

Provide logos at different sizes for various contexts:

```yaml
logo:
  # Favicon and small contexts
  small: "brand/favicon.png"

  # Navigation bars, headers
  medium: "brand/logo-nav.png"

  # Hero sections, print materials
  large: "brand/logo-full.png"
```

Logo usage across formats.

Format	Location	Logo Preference (high to low)
`html` / `dashboard`	Top navigation bar	`small` > `medium` > `large`
`html`	Side navigation	`medium` > `small` > `large`
`typst`	Top left, control with `format: typst: logo`	`medium` > `small` > `large`
`revealjs`	Bottom right corner of slides	`medium` > `small` > `large`
`website` / `book` project	`favicon` shown in browser tab	`small`

See Logo Configuration for more.

Brand Integration

Combine `brand.yml` with Bootstrap customisation:

_quarto.yml

```yaml
format:
  html:
    theme:
      - cosmo
      - brand
      - styles.scss
    css: custom-components.css
```

1

Brand Integration

Combine `brand.yml` with Bootstrap customisation:

_quarto.yml

```yaml
format:
  html:
    theme:
      - cosmo
      - brand
      - styles.scss
    css: custom-components.css
```

1

Note

The brand colours become available as CSS variables and SCSS variables automatically.

Brand Integration

Combine `brand.yml` with Bootstrap customisation:

_quarto.yml

```yaml
format:
  html:
    theme:
      - cosmo
      - brand
      - styles.scss
    css: custom-components.css
```

Prefer `brand.yml` or SCSS for component customisation. [1]

Note

The brand colours become available as CSS variables and SCSS variables automatically.

Brand Integration

Combine `brand.yml` with Bootstrap customisation:

_quarto.yml

```yaml
format:
  html:
    theme:
      - cosmo
      - brand
      - styles.scss
    css: custom-components.css
```

1

Note

The brand colours become available as CSS variables and SCSS variables automatically.

- What Are Brand Extensions?
- Creating a Brand Extension

Brand Extensions Development

 |

What Are Brand Extensions?

 |

What Are Brand Extensions?

Brand extensions are Quarto extensions that provide a `brand.yml` file and its assets, packaged for easy distribution and reuse.

What Are Brand Extensions?

Brand extensions are Quarto extensions that provide a `brand.yml` file and its assets, packaged for easy distribution and reuse.

Key features:

What Are Brand Extensions?

Brand extensions are Quarto extensions that provide a `brand.yml` file and its assets, packaged for easy distribution and reuse.

Key features:

- **Portable branding** - Install everywhere.

What Are Brand Extensions?

Brand extensions are Quarto extensions that provide a `brand.yml` file and its assets, packaged for easy distribution and reuse.

Key features:

- **Portable branding** - Install everywhere.
- **Team consistency** - Everyone gets the same brand.

 |

What Are Brand Extensions?

Brand extensions are Quarto extensions that provide a `brand.yml` file and its assets, packaged for easy distribution and reuse.

Key features:

- **Portable branding** - Install everywhere.
- **Team consistency** - Everyone gets the same brand.
- **Asset management** - Logos and fonts included.

What Are Brand Extensions?

Brand extensions are Quarto extensions that provide a `brand.yml` file and its assets, packaged for easy distribution and reuse.

Key features:

- **Portable branding** - Install everywhere.
- **Team consistency** - Everyone gets the same brand.
- **Asset management** - Logos and fonts included.
- **Version control** - Track brand changes over time.

Creating a Brand Extension

 |

Creating a Brand Extension

Step 1: Scaffold Extension

BASH

```bash
quarto create extension brand my-brand
cd my-brand
```

 |

Creating a Brand Extension

Step 1: Scaffold Extension

BASH

```bash
quarto create extension brand my-brand
cd my-brand
```

This creates:

```
my-brand
├── README.md
├── _extensions
│   └── my-brand
│       ├── _extension.yml
│       └── brand.yml
├── _quarto.yml
└── example.qmd

2 directories, 5 files
```

Creating a Brand Extension

Step 2: Configure Extension

Quarto

```yaml
title: My-brand
author: First Last
version: 1.0.0
quarto-required: "≥1.9.0"
contributes:
  metadata:
    project:
      brand: brand.yml
```

 |

Creating a Brand Extension

Step 3: Design Brand System

_extensions/my-brand/brand.yml

```yaml
meta:
  name: "My Organisation"
  link: "https://myorg.com"

color:
  palette:
    # Primary brand colours
    brand-blue: "#2E86AB"
    brand-coral: "#A23B72"
    brand-gold: "#F4A261"
    # Supporting colours
    neutral-light: "#F8F9FA"
    neutral-dark: "#343A40"
    success-green: "#28A745"
    warning-amber: "#FFC107"
  # Semantic mappings
  primary: brand-blue
  secondary: brand-coral
  success: success-green
  warning: warning-amber
  foreground: neutral-dark
  background: neutral-light

logo:
  small: assets/logo/favicon.png
  medium: assets/logo/logo-horizontal.png
  large: assets/logo/logo-vertical.png

typography:
  fonts:
    - family: "Source Sans Pro"
      source: "google"
      weight: [300, 400, 600, 700]

  base: "Source Sans Pro"
```

_extensions/my-brand/brand.yml

```yaml
defaults:
  bootstrap:
    defaults:
      # Enhanced navigation
      navbar-padding-y: 1rem
      navbar-brand-font-size: 1.5rem

      # Improved buttons
      btn-padding-y: 0.5rem
      btn-padding-x: 1.5rem
      btn-font-weight: 600

      # Better spacing
      spacer: 1rem

    rules: |
      /* Professional enhancements */
      .navbar-brand {
        font-weight: 700;
      }

      .btn-primary {
        border-radius: 2rem;
        transition: all 0.3s ease;
      }

      .btn-primary:hover {
        transform: translateY(-1px);
      }

      /* Content styling */
      .title-block-header {
        padding: 3rem 2rem;
        margin-bottom: 2rem;
        border-radius: 0.5rem;
```

Creating a Brand Extension

Step 4: Add Assets

Organise brand assets:

TXT

```
_extensions/my-brand/
├── _extension.yml
├── brand.yml
└── assets/
    ├── logo/
    │   ├── favicon.png
    │   ├── logo-horizontal.png
    │   └── logo-vertical.png
    └── fonts/
        └── custom-font.woff2
```

Creating a Brand Extension

Step 5: Create Example

example.qmd

```yaml
---
title: "Brand Extension Showcase"
subtitle: "Demonstrating consistent branding"
author: "Your Name"
format:
  html:
    theme:
      - quartz
      - brand
---

# Welcome to Our Brand System

This document demonstrates the power of brand extensions in Quarto.

[Primary Action]{.btn .btn-primary}
[Secondary Action]{.btn .btn-secondary}
```

Important

Brand extensions require a `_quarto.yml` project file to work.

- Exercise Overview
- Part 1: Brand Extension Setup
- Part 2: Testing
- Success Criteria

Hands-On Exercise: Building Your Brand Extension

Exercise Overview

Objective: Create a comprehensive brand extension that demonstrates unified branding across formats using `brand.yml` and light CSS customisation.

Example Code: Exercises Brand

BASH

```bash
tar -xzf "05a-exercises.tar.gz" -C "05-formatting-branding"
```

Part 1: Brand Extension Setup

1. **Generate extension scaffold**:

```bash
quarto create extension brand my-company-brand
cd my-company-brand
```

2. **Define your brand identity.**

3. **Add styling.**

 |

Part 2: Testing

1. **Build test project**:

```bash
quarto create project
```

2. **Test across formats**:

```bash
# Test HTML output
quarto render example.qmd --to html

# Test presentation
quarto render example.qmd --to revealjs

# Test Typst output
quarto render example.qmd --to typst
```

3. **Verify brand consistency.**

Success Criteria

You've successfully completed the exercise if you can:

- Create a working brand extension with consistent visual identity.
- Implement styling using `brand.yml` and light (S)CSS.
- Test the brand system across multiple formats.
- Demonstrate cross-format consistency.
- Build a reusable brand asset for teams.
- Apply design principles effectively.

 |

- Understanding Pandoc Templates
- Basic Variable Syntax
- Conditional Content
- Loops for Multiple Items
- Separator Usage

Pandoc Templating System

 |

Understanding Pandoc Templates

 |

Understanding Pandoc Templates

Quarto uses Pandoc’s templating system to generate output.

Understanding Pandoc Templates

Quarto uses Pandoc’s templating system to generate output.

Key concepts:

 |

Understanding Pandoc Templates

Quarto uses Pandoc's templating system to generate output.

Key concepts:

- **Variables** - `$variable$` inserts YAML metadata.

 |

Understanding Pandoc Templates

Quarto uses Pandoc's templating system to generate output.

Key concepts:

- **Variables** - `$variable$` inserts YAML metadata.
- **Conditionals** - `$if(variable)$ ... $endif$` for optional content.

 |

Understanding Pandoc Templates

Quarto uses Pandoc's templating system to generate output.

Key concepts:

- **Variables** - `$variable$` inserts YAML metadata.
- **Conditionals** - `$if(variable)$ ... $endif$` for optional content.
- **Loops** - `$for(variable)$ ... $endfor$` for repeated content.

 |

Understanding Pandoc Templates

Quarto uses Pandoc's templating system to generate output.

Key concepts:

- **Variables** - `$variable$` inserts YAML metadata.
- **Conditionals** - `$if(variable)$ ... $endif$` for optional content.
- **Loops** - `$for(variable)$ ... $endfor$` for repeated content.
- **Partials** - Reusable template components.

 |

Understanding Pandoc Templates

Quarto uses Pandoc's templating system to generate output.

Key concepts:

- **Variables** - `$variable$` inserts YAML metadata.
- **Conditionals** - `$if(variable)$ ... $endif$` for optional content.
- **Loops** - `$for(variable)$ ... $endfor$` for repeated content.
- **Partials** - Reusable template components.

See Pandoc Template Syntax for complete documentation.

Basic Variable Syntax

 |

Basic Variable Syntax

Variables from YAML front matter are inserted using `$variable$`:

 |

Basic Variable Syntax

Variables from YAML front matter are inserted using `$variable$`:

Quarto Document

```yaml
---
title: "My Report"
author: "Jane Smith"
date: "2025-01-15"
---
```

Template Input

```
Title: $title$
Author: $author$
Date: $date$
```

Template Output

```
Title: My Report
Author: Jane Smith
Date: 2025-01-15
```

Basic Variable Syntax

Variables from YAML front matter are inserted using `$variable$`:

Quarto Document

```yaml
---
title: "My Report"
author: "Jane Smith"
date: "2025-01-15"
---
```

Template Input

```
Title: $title$
Author: $author$
Date: $date$
```

Template Output

```
Title: My Report
Author: Jane Smith
Date: 2025-01-15
```

Tip

Use `$variable$` in any template file (`.typ`, `.tex`, `.html`, etc.).

Basic Variable Syntax

Variables from YAML front matter are inserted using `$variable$`:

Quarto Document

```yaml
---
title: "My Report"
author: "Jane Smith"
date: "2025-01-15"
---
```

Template Input

```
Title: $title$
Author: $author$
Date: $date$
```

Template Output

```
Title: My Report
Author: Jane Smith
Date: 2025-01-15
```

Tip

Use `$variable$` in any template file (`.typ`, `.tex`, `.html`, etc.).

See Article Templates for templates and partials.

Conditional Content

 |

Conditional Content

Show content only when a variable exists:

 |

Conditional Content

Show content only when a variable exists:

Quarto Document

```yaml
---
title: "Report"
subtitle: "Q4 Analysis"
# No author specified
---
```

Template Input

```
Title: $title$

$if(subtitle)$
Subtitle: $subtitle$
$endif$

$if(author)$
Author: $author$
$endif$
```

Template Output

```
Title: Report
Subtitle: Q4 Analysis
```

Conditional Content

Show content only when a variable exists:

Quarto Document

```yaml
---
title: "Report"
subtitle: "Q4 Analysis"
# No author specified
---
```

> **Note**
>
> Use `$if(variable)$` ... `$else$` ... `$endif$` for alternative content.

Template Input

```
Title: $title$

$if(subtitle)$
Subtitle: $subtitle$
$endif$

$if(author)$
Author: $author$
$endif$
```

Template Output

```
Title: Report
Subtitle: Q4 Analysis
```

Conditional Content

Show content only when a variable exists:

Quarto Document

```yaml
---
title: "Report"
subtitle: "Q4 Analysis"
# No author specified
---
```

Template Input

```
Title: $title$

$if(subtitle)$
Subtitle: $subtitle$
$endif$

$if(author)$
Author: $author$
$endif$
```

Template Output

```
Title: Report
Subtitle: Q4 Analysis
```

Note

Use `$if(variable)$` ... `$else$` ... `$endif$` for alternative content.

Important

`$if(variable)$` in Pandoc templates checks if a variable is:

- Defined (exists in the metadata).
- Non-empty (has a value).
- Not explicitly false.

Loops for Multiple Items

 |

Loops for Multiple Items

Process lists from YAML metadata:

 |

Loops for Multiple Items

Process lists from YAML metadata:

Quarto Document

```yaml
---
authors:
  - name: Jane Smith
    email: jane@example.com
  - name: John Doe
    email: john@example.com
---
```

Template Input

```
Authors:
$for(authors)$
- $it.name$ ($it.email$)
$endfor$
```

Template Output

```
Authors:
- Jane Smith (jane@example.com)
- John Doe (john@example.com)
```

Loops for Multiple Items

Process lists from YAML metadata:

Quarto Document

```yaml
---
authors:
  - name: Jane Smith
    email: jane@example.com
  - name: John Doe
    email: john@example.com
---
```

Template Input

```
Authors:
$for(authors)$
- $it.name$ ($it.email$)
$endfor$
```

Template Output

```
Authors:
- Jane Smith (jane@example.com)
- John Doe (john@example.com)
```

Note

Use `$it$` to reference the current item in a loop. Use `$sep$` followed by the separator string for separators.

Separator Usage

Add separators between items:

Quarto Document

```yaml
---
tags:
  - Research
  - Data Science
  - Machine Learning
---
```

Template Input

```
Tags: $for(tags)$$it$$sep$, $endfor$
```

Template Output

```
Tags: Research, Data Science, Machine Learning
```

Template Input

```
Tags: $for(tags)$$it$$endfor$
```

Template Output

```
Tags: ResearchData ScienceMachine Learning
```

 |

- Exercise Overview
- Part 1: Template Modification
- Part 2: Customise Further
- Success Criteria

Hands-On Exercise: Pandoc Templating

 |

Exercise Overview

Objective: Modify a partial template to display additional metadata using Pandoc templating syntax.

Example Code: Exercises Partials

BASH

```bash
tar -xzf "05b-exercises.tar.gz" -C "05-custom-partials"
```

Part 1: Template Modification

1. **Retrieve the partials** described in Article Templates
   - Typst Partials
   - HTML Partials
   - Reveal.js Partials
2. **Modify the title slide partial from Reveal.js** to underline the corresponding author:
   - Use `$if(by-author)$` to check for `by-author` metadata.
   - Loop through `by-author` to find the corresponding author and underline their name.
   - See Author Schema for details.

Part 2: Customise Further

1. **Add additional metadata** to the title slide, such as affiliations or ORCID IDs.
2. **Test the changes** by rendering a sample presentation with multiple authors with affiliations and a corresponding author.

 |

Success Criteria

You've successfully completed the exercise if you can:

- Modify a Quarto/Pandoc partial template using conditionals and loops.
- Display dynamic content based on YAML metadata.
- Test and verify the output in a Reveal.js presentation.

 |

- Typst in Quarto
- Typst Template Structure
- Simple Customisation Example
- Creating a Typst Format

Typst Customisation (For the Curious)

 |

Typst in Quarto

 |

Typst in Quarto

Typst is a modern typesetting system for creating beautiful PDFs, designed to be simpler than LaTeX.

Typst in Quarto

Typst is a modern typesetting system for creating beautiful PDFs, designed to be simpler than LaTeX.

Why Typst?

Typst in Quarto

Typst is a modern typesetting system for creating beautiful PDFs, designed to be simpler than LaTeX.

Why Typst?

- **Fast** - Instant PDF generation.

 |

Typst in Quarto

Typst is a modern typesetting system for creating beautiful PDFs, designed to be simpler than LaTeX.

Why Typst?

- **Fast** - Instant PDF generation.
- **Simple** - Markdown-like syntax.

 |

Typst in Quarto

Typst is a modern typesetting system for creating beautiful PDFs, designed to be simpler than LaTeX.

Why Typst?

- **Fast** - Instant PDF generation.
- **Simple** - Markdown-like syntax.
- **Flexible** - Easy template customisation.

Typst in Quarto

Typst is a modern typesetting system for creating beautiful PDFs, designed to be simpler than LaTeX.

Why Typst?

- **Fast** - Instant PDF generation.
- **Simple** - Markdown-like syntax.
- **Flexible** - Easy template customisation.

See Typst Basics for more.

Typst Template Structure

 |

Typst Template Structure

Quarto's Typst templates use two key files:

 |

Typst Template Structure

Quarto's Typst templates use two key files:

- `typst-template.typ` - Defines document layout:

typst-template.typ

```
#let article(title: none, authors: none, body) = {
  set page(paper: "a4", margin: 2.5cm)                    (1)

  set text(font: "Inter", size: 11pt)                     (2)

  if title ≠ none {                                       (3)
    align(center, text(weight: "bold", 1.5em, title))
  }

  body                                                    (4)
}
```

Typst Template Structure

Quarto's Typst templates use two key files:

- `typst-template.typ` - Defines document layout:

typst-template.typ

```
1  #let article(title: none, authors: none, body) = {
2    set page(paper: "a4", margin: 2.5cm)
3
4    set text(font: "Inter", size: 11pt)
5
6    if title ≠ none {
7      align(center, text(weight: "bold", 1.5em, title))
8    }
9
10   body
11 }
```

(1) Page setup.

(2)

(3)

(4)

Typst Template Structure

Quarto's Typst templates use two key files:

- `typst-template.typ` - Defines document layout:

typst-template.typ

```
1  #let article(title: none, authors: none, body) = {
2    set page(paper: "a4", margin: 2.5cm)
3
4    set text(font: "Inter", size: 11pt)
5
6    if title ≠ none {
7      align(center, text(weight: "bold", 1.5em, title))
8    }
9
10   body
11 }
```

(1) (2) (3) (4)

(2) Typography settings.

Typst Template Structure

Quarto's Typst templates use two key files:

- `typst-template.typ` - Defines document layout:

typst-template.typ

```
1  #let article(title: none, authors: none, body) = {
2    set page(paper: "a4", margin: 2.5cm)
3
4    set text(font: "Inter", size: 11pt)
5
6    if title ≠ none {
7      align(center, text(weight: "bold", 1.5em, title))
8    }
9
10   body
11 }
```

(1) (2) (3) Conditional title bloc (4)

Typst Template Structure

Quarto's Typst templates use two key files:

- `typst-template.typ` - Defines document layout:

typst-template.typ

```
1  #let article(title: none, authors: none, body) = {
2    set page(paper: "a4", margin: 2.5cm)                    (1)
3
4    set text(font: "Inter", size: 11pt)                     (2)
5
6    if title ≠ none {                                       (3)
7      align(center, text(weight: "bold", 1.5em, title))
8    }
9
10   body                                                     (4)
11 }
```

(4) Main content insertio

Typst Template Structure

 |

Typst Template Structure

- `typst-show.typ` - Captures YAML metadata using Pandoc syntax:

typst-show.typ

```
#show: doc ⇒ article(
  $if(title)$
  title: [$title$],
  $endif$
  $if(by-author)$
  authors: [$for(by-author)$$it.name.literal$$sep$, $endfor$],
  $endif$
  doc,
)
```

(1)

Typst Template Structure

- `typst-show.typ` - Captures YAML metadata using Pandoc syntax:

typst-show.typ

```
#show: doc ⇒ article(
  $if(title)$
  title: [$title$],
  $endif$
  $if(by-author)$
  authors: [$for(by-author)$$it.name.literal$$sep$, $endfor$],
  $endif$
  doc,
)
```

(1)

> **Note**
>
> Notice the Pandoc template syntax: `$if()$`, `$for()$`, `$variable$`.

Typst Template Structure

- `typst-show.typ` - Captures YAML metadata using Pandoc syntax:

typst-show.typ

```
1  #show: doc ⇒ article(
2    $if(title)$
3    title: [$title$],
4    $endif$
5    $if(by-author)$
6    authors: [$for(by-author)$$it.name.literal$$sep$, $endfor$],
7    $endif$
8    doc,
9  )
```

Use `by-author` to get Quarto's extended author metadata. (1)

> **Note**
>
> Notice the Pandoc template syntax: `$if()$`, `$for()$`, `$variable$`.

Simple Customisation Example

typst-template.typ

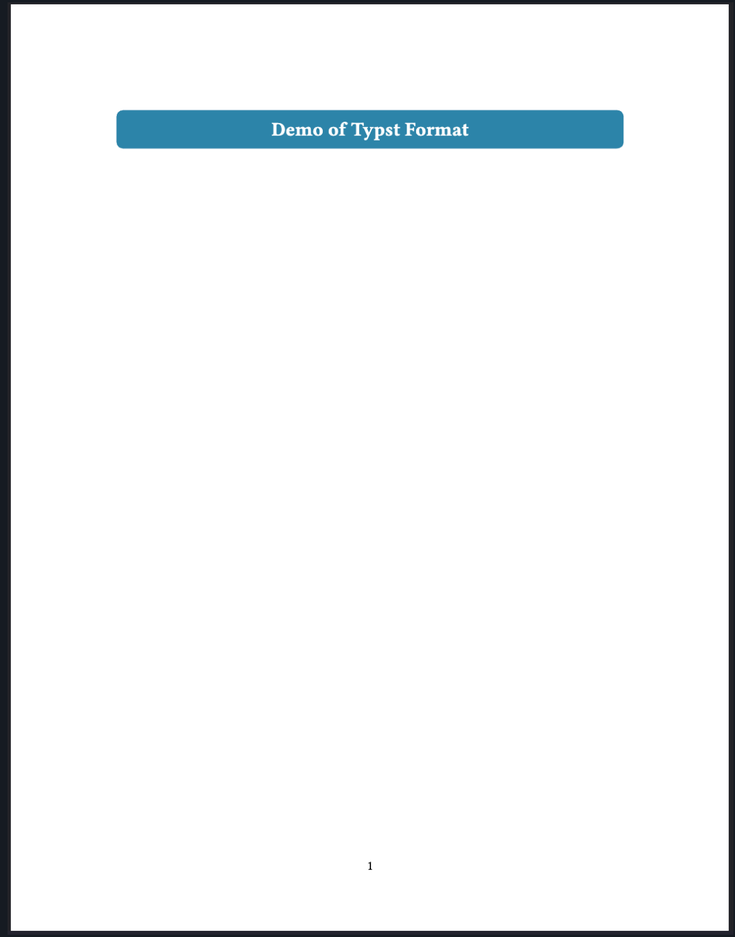
```typst
#let article(
  title: none,
  body
) = {
  let primary = rgb("#2E86AB")

  if title ≠ none {
    block(
      fill: primary,
      inset: 1em,
      radius: 0.5em,
      width: 100%
    )[
      #align(center)[
        #text(
          fill: white,
          weight: "bold",
          size: 1.5em,
          title
        )
      ]
    ]
  }

  body
}
```

typst-show.typ

```typst
#show: doc ⇒ article(
  $if(title)$
  title: [$title$],
  $endif$
  doc,
)
```



Screenshot of a PDF document with blue header titled Demo of Typst Format on white background with page number 1 at bottom

Creating a Typst Format

 |

Creating a Typst Format

Step 1: Scaffold Extension

BASH

```bash
quarto create extension format:typst my-format
cd my-format
```

 |

Creating a Typst Format

Step 1: Scaffold Extension

BASH

```bash
quarto create extension format:typst my-format
cd my-format
```

This creates:

```
my-format
├── README.md
├── _extensions
│   └── my-format
│       ├── _extension.yml
│       ├── typst-show.typ
│       └── typst-template.typ
└── template.qmd

2 directories, 5 files
```

Creating a Typst Format

Step 1: Scaffold Extension

```bash
quarto create extension format:typst my-format
cd my-format
```

This creates:

```
my-format
├── README.md
├── _extensions
│   └── my-format
│       ├── _extension.yml
│       ├── typst-show.typ
│       └── typst-template.typ
└── template.qmd

2 directories, 5 files
```

> **Tip**
>
> Modify the generated `typst-template.typ` file to customise your layout.

Creating a Typst Format

Step 2: Integration with `brand.yml`

_brand.yml

```yaml
color:
  primary: "#2E86AB"

logo:
  medium: "assets/logo.png"
```

typst-show.typ

```
#show: doc => article(
  $if(title)$
  title: [$title$],
  title_colour: brand-color.primary,
  $endif$
  doc,
)
```

page.typ

```
$if(logo)$
#set page(
  header: context {
    if counter(page).get().first() == 1{
      place(top + left, dy: 0.5cm, image("$logo.path$", width: 2cm))
    }
  }
)
$endif$
```

typst-template.typ

```
#let article(
  title: none,
  title_colour: "#000000",
  body
) = {
  set text(size: 11pt)
  if title =/= none {
    align(center)[
      #text(
        fill: rgb(title_colour),
        weight: "bold",
        size: 1.5em,
        title
      )
    ]
  }
  body
}
```

- Exercise Overview
- Part 1: Format Extension Setup
- Part 2: Customise and Test Your Format
- Part 3: Optional HTML Variant
- Success Criteria

Hands-On Exercise: Creating a Format Extension

 |

Exercise Overview

Objective: Create a custom format extension that bundles `brand.yml` and styling into a reusable format for your organisation.

Example Code: Exercises Typst

BASH

```bash
tar -xzf "05c-exercises.tar.gz" -C "05-format-extension"
```

 |

Part 1: Format Extension Setup

1. **Generate format extension scaffold**:

BASH

```bash
quarto create extension format:html company-report
cd company-report
```

2. **Add brand.yml and configure extension** in _extension.yml`:

YAML

```yaml
contributes:
  formats:
    typst:
      template-partials:
        - assets-typst/typst-template.typ
        - assets-typst/typst-show.typ
        - assets-typst/page.typ
  metadata:
    project:
      brand: brand.yml
```

Part 2: Customise and Test Your Format

1. **Define your template** in `typst-template.typ` and `typst-show.typ`.
2. **Use your format** in `template.qmd`:

```yaml
---
title: "Quarterly Report"
format: company-report-typst
---
```

3. **Render and verify**:

```bash
quarto render template.qmd
```

Part 3: Optional HTML Variant

Add HTML output to your format:

1. **Add HTML format** to `_extension.yml`.
2. **Create/Modify simple template**. (See HTML partials)
3. **Test both formats**:

```bash
quarto render template.qmd --to company-report-html
quarto render template.qmd --to company-report-typst
```

Success Criteria

You've successfully completed the exercise if you can:

- Create a working format extension with bundled branding.
- Apply custom styling through brand.yml and SCSS.
- Test the format with sample documents.
- **(Optional)** Add HTML variant.

 |

Mickaël CANOUIL, *Ph.D.*

Independent Contractor

pro@mickael.canouil.dev

Website
mickael.canouil.fr

GitHub
@mcanouil

LinkedIn
MickaelCanouil

Bluesky
@mickael.canouil.fr

Mastodon
@MickaelCanouil

 |