Lecture on Quarto

Documents

Irene Spada

Introduzione alla Data Science per Ingegneria

Corso di Laurea Triennale di Ingegneria Gestionale

Univeristà di Pisa

2025-03-15

Anatomy of a Quarto document

Components

1. Metadata: YAML
2. Text: Markdown
3. Code: Executed via `knitr` or `jupyter`

Weave it all together, and you have beautiful, powerful, and useful outputs!

Literate programming

Literate programming is writing out the program logic in a human language with included (separated by a primitive markup) code snippets and macros.

~~~
---
title: "ggplot2 demo"
date: "6/6/2023"
format: html
---

## MPG

There is a relationship between city and highway mileage.

```{r}
#| label: fig-mpg

library(ggplot2)

ggplot(mpg, aes(x = cty, y = hwy)) +
  geom_point() +
  geom_smooth(method = "loess")
```
~~~

Metadata

YAML

“Yet Another Markup Language” or “YAML Ain’t Markup Language” is used to provide document level metadata.

```
---
key: value
---
```

Output options

```yaml
---
format: something
---
```

```yaml
---
format: html
---
```

```yaml
---
format: pdf
---
```

```yaml
---
format: revealjs
---
```

Output option arguments

Indentation matters!

```yaml
---
format:
  html:
    toc: true
    code-fold: true
---
```

YAML validation

- Invalid: No space after `:`

```yaml
---
format:html
---
```

- Invalid: Read as missing

```yaml
---
format:
html
---
```

- Valid, but needs next object

```yaml
---
format:
  html:
---
```

YAML validation

There are multiple ways of formatting valid YAML:

- Valid: There's a space after `:`

```yaml
format: html
```

- Valid: There are 2 spaces a new line and no trailing `:`

```yaml
format:
  html
```

- Valid: `format: html` with selections made with proper indentation

```yaml
format:
  html:
    toc: true
```

Why YAML?

To avoid manually typing out all the options, every time when rendering via the CLI:

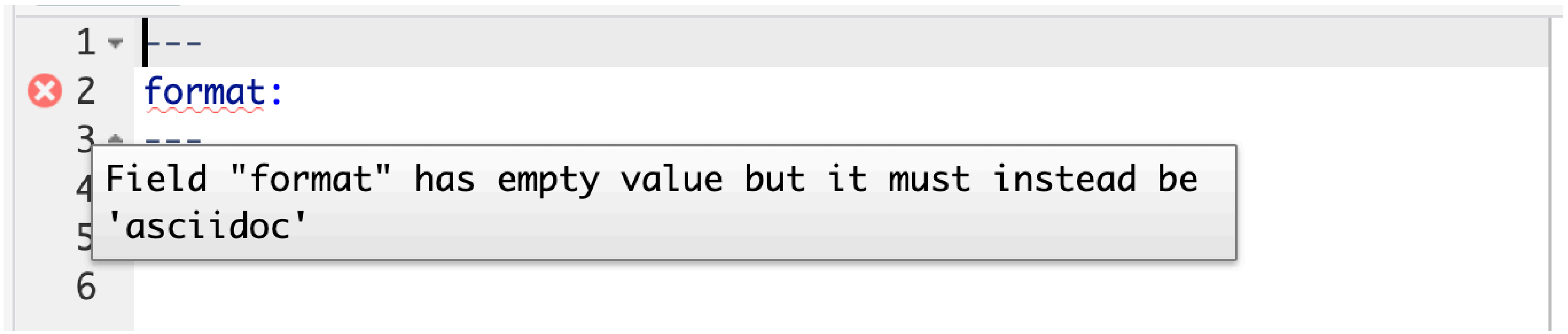
```
quarto render document.qmd --to html
```

```
quarto render document.qmd --to html -M code-fold:true
```

```
quarto render document.qmd --to html -M code-fold:true -P alpha:0.2 -P ratio:0.3
```

Quarto linting

Lint, or a linter, is a static code analysis tool used to flag programming errors, bugs, stylistic errors and suspicious constructs.



```
---
format:
---
```

```
Field "format" has empty value but it must instead be
'asciidoc'
```

Quarto YAML Intelligence

RStudio + VSCode provide rich tab-completion - start a word and tab to complete, or `Ctrl + space` to see all available options.

Your turn

- Open `hello-penguins.qmd` in RStudio.
- Try `Ctrl + space` to see the available document YAML options.
- Try out the tab-completion of any options you remember.
- You can use the HTML reference as needed.

05:00

List of valid YAML fields

- Many YAML fields are common across various outputs
- But also each output type has its own set of valid YAML fields and options
- Definitive list: quarto.org/docs/reference/formats/html

Text

Text Formatting

Markdown Syntax	Output
`*italics* and **bold**`	*italics* and **bold**
`superscript^2^ / subscript~2~`	superscript^2 / subscript$_2$
`~~strikethrough~~`	~~strikethrough~~
`` `verbatim code` ``	`verbatim code`

Headings

Markdown Syntax	Output
`# Header 1`	Header 1
`## Header 2`	Header 2
`### Header 3`	Header 3
`#### Header 4`	Header 4
`##### Header 5`	Header 5
`###### Header 6`	Header 6

Links

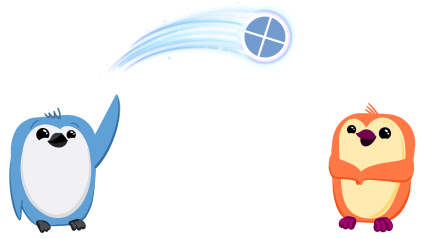
There are several types of "links" or hyperlinks.

Markdown

```
You can embed [named hyperlinks](https://quarto.org/),
direct urls like <https://quarto.org/>, and links to
[other places](#quarto-anatomy) in
the document. The syntax is similar for embedding an
inline image: ![Penguins playing with ball](images/penguins-quarto-b
```

Output

You can embed named hyperlinks, direct urls like https://quarto.org/, and links to other places in the document. The syntax is similar for embedding an inline image:



.

Lists

Unordered list:

Markdown:

```
-   unordered list
    -   sub-item 1
    -   sub-item 1
        -   sub-sub-item 1
```

Output

- unordered list
    - sub-item 1
    - sub-item 1
        - sub-sub-item 1

Ordered list:

Quotes

Markdown:

```
> Let us change our traditional attitude to the construction of programs: Instead
> - Donald Knuth, Literate Programming
```

Output:

> Let us change our traditional attitude to the construction of programs: Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to human beings what we want a computer to do. - Donald Knuth, Literate Programming

“Literate Programming”, The Computer Journal 27 (1984), p. 97. (Reprinted in Literate

Divs and spans

Pandoc, and therefore Quarto, can parse "fenced div blocks":

- You can think of a `:::` **div** as a HTML `<div>` but it can also apply in specific situations to content in PDF:

```
::: {style="border-left:10px solid red"}
This content can be styled with a border
:::
```

This content can be styled with a border

- `[text]{.class}` **spans** can be thought of a `<span .class>Text</span>` but again are a bit more transferable if using Pandoc/Quarto native attributes.

```
This is text with [special]{style="color:red;"} formatting.
```

This is text with special formatting.

Your turn

- Open `markdown-syntax.qmd` in RStudio.
- Follow the instructions in the document, then exchange one new thing you've learned with your neighbor.

05:00

Divs with pre-defined classes

These can often apply between formats:

```
::: {.unnumbered .unlisted}
Text
:::
```

Callouts

```
::: callout-note
Note that there are five types of callouts, including:
`note`, `tip`, `warning`, `caution`, and `important`.
:::
```

Note

Note that there are five types of callouts, including: `note`, `tip`, `warning`, `caution`, and `important`.

More callouts

Warning

Callouts provide a simple way to attract attention, for example, to this warning.

Important

Danger, callouts will really improve your writing.

Caution

Here is something under construction.

Caption

Tip with caption.

Your turn

- Open `callout-boxes.qmd` and render the document.
- Using the visual editor, change the type of the first callouts box and then re-render. Also play with the options to change its appearance.
- Add a caption to the second callout box.
- Make the third callout box collapsible. Then, switch over to the source editor to inspect the markdown code.
- Change the format to PDF and re-render.

05:00

Footnotes

Pandoc supports numbering and formatting footnotes.

Inline footnotes

```
Here is an inline note.^[Inlines notes are easier to write,
since you don't have to pick an identifier and move down to
type the note.]
```

Here is an inline note.[1]

Inline footnotes

```
Here is an footnore reference[^1]

[^1]: This can be easy in some situations when you have a really long note or
don't want to inline complex outputs.
```

Here is an footnote reference[1]

Notice in both situations that the footnote is placed at the bottom of the page in presentations, whereas in a document it would be hoverable or at the end of the document.

Code

Anatomy of a code chunk

~~~
```{r}
#| label: car-stuff
#| message: false

library(tidyverse)
library(palmerpenguins)

penguins |>
  distinct(species)
```
~~~

```
# A tibble: 3 × 1
  species
  <fct>
1 Adelie
2 Gentoo
3 Chinstrap
```

- Has 3x backticks on each end
- Engine (`r`) is indicated between curly braces `{r}`
- Options stated with the `#|` (hashpipe): `#| option1: value`

Code, who is it for?

- The way you display code is very different for different contexts.
- In a teaching scenario like today, I *really* want to display code.
- In a business, you may want to have a data-science facing output which displays the source code AND a stakeholder-facing output which hides the code.

Code

If you simply want code formatting but don’t want to execute the code:

- Option 1: Use 3x back ticks + the language ` ```r `

~~~
```r
head(penguins)
```
~~~

- Option 2: Add `eval: false` as chunk option

~~~
```{r}
#| eval: false

head(penguins)
```
~~~

Showing and hiding code with `echo`

- The `echo` option shows the code when set to `true` and hides it when set to `false`.
- If you want to both execute the code and return the full code including backticks (like in a teaching scenario) `echo: fenced` is your friend!

~~~
```{r}
1 + 1
```
~~~

```
[1] 2
```

Tables and figures

- In reproducible reports and manuscripts, the most commonly included code outputs are **tables** and **figures**.
- So they get their own special sections in our deep dive!

Tables

Markdown tables

Markdown:

```
| Right | Left | Default | Center |
|------:|:-----|---------|:------:|
|   12  |  12  |    12   |    12  |
|  123  |  123 |   123   |   123  |
|    1  |    1 |     1   |     1  |
```

Output:

Right	Left	Default	Center
12	12	12	12
123	123	123	123
1	1	1	1

Grid tables

Markdown:

```
+---------------+---------------+--------------------+
| Fruit         | Price         | Advantages         |
+===============+===============+====================+
| Bananas       | $1.34         | - built-in wrapper |
|               |               | - bright color     |
+---------------+---------------+--------------------+
| Oranges       | $2.10         | - cures scurvy     |
|               |               | - tasty            |
+---------------+---------------+--------------------+

: Sample grid table.
```

Grid tables

Output:

Sample grid table.

Fruit	**Price**	**Advantages**
Bananas	$1.34	• built-in wrapper • bright color

Grid tables: Alignment

- Alignments can be specified as with pipe tables, by putting colons at the boundaries of the separator line after the header:

```
+---------------+---------------+--------------------+
| Right         | Left          | Centered           |
+==============:+:==============+:==================:+
| Bananas       | $1.34         | built-in wrapper   |
+---------------+---------------+--------------------+
```

- For headerless tables, the colons go on the top line instead:

```
+--------------:+:--------------+:------------------:+
| Right         | Left          | Centered           |
+---------------+---------------+--------------------+
```

Grid tables: Authoring

- Note that grid tables are quite awkward to write with a plain text editor because unlike pipe tables, the column indicators must align.
- The Visual Editor can assist in making these tables!

Tables from code

The **knitr** package can turn data frames into tables with `knitr::kable()`:

```r
library(knitr)

head(penguins) |>
  kable()
```

species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_r
Adelie	Torgersen	39.1	18.7	181	
Adelie	Torgersen	39.5	17.4	186	
Adelie	Torgersen	40.3	18.0	195	
Adelie	Torgersen	NA	NA	NA	
Adelie	Torgersen	36.7	19.3	193	
Adelie	Torgersen	39.3	20.6	190	

Tables from code

If you want fancier tables, try the **gt** package and all that it offers!

```r
library(gt)

head(penguins) |>
  gt() |>
  tab_style(
    style = list(
      cell_fill(color = "pink"),
      cell_text(style = "italic")
      ),
    locations = cells_body(
      columns = bill_length_mm,
      rows = bill_length_mm > 40
    )
  )
```

species	island	bill_length_mm	bill_depth_mm	flipper_length_mm
Adelie	Torgersen	39.1	18.7	181
Adelie	Torgersen	39.5	17.4	186
Adelie	Torgersen	*40.3*	18.0	195
Adelie	Torgersen	NA	NA	NA
Adelie	Torgersen	36.7	19.3	193
Adelie	Torgersen	39.3	20.6	190

Figures

Markdown figures

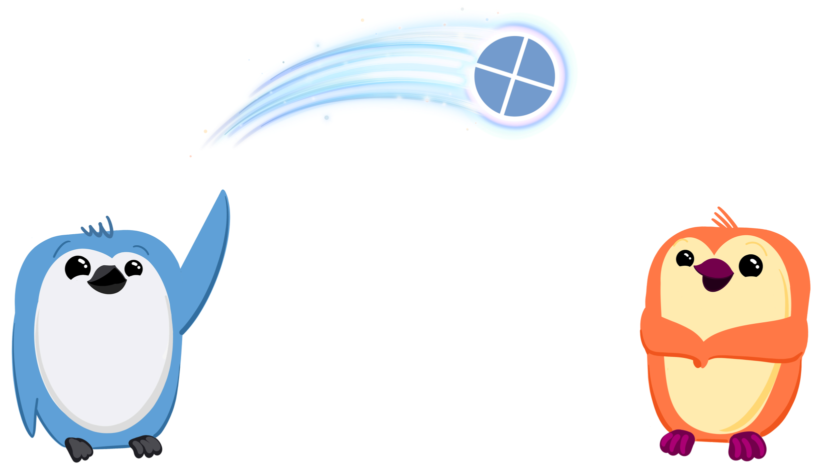
```
![Penguins playing with a Quarto ball](images/penguins-quarto-ball.png)
```

Penguins playing with a Quarto ball

Markdown figures with options

```
![Penguins playing with a Quarto ball](images/penguins-quarto-ball.png){fig-align="left"}
```

```
![](images/penguins-quarto-ball.png){fig-align="right" fig-alt="Illustration of two penguins playing with a Quarto ball."}
```



Subfigures

Markdown:

```
::: {#fig-penguins layout-ncol=2}

![Blue penguin](images/blue-penguin.png){#fig-blue width="250px"}

![Orange penguin](images/orange-penguin.png){#fig-orange width="250px"}

Two penguins

:::
```

Subfigures

Output:

(a) Blue penguin

(b) Orange penguin

Figure 1: Two penguins

Figure divs

Markdown:

```
::: {#fig-penguin}

<iframe width="560" height="315" src="https://www.youtube.com/embed/q3uXXh1sHcI"><

Baby penguin tries to make friends
:::
```

Figure divs

Output:

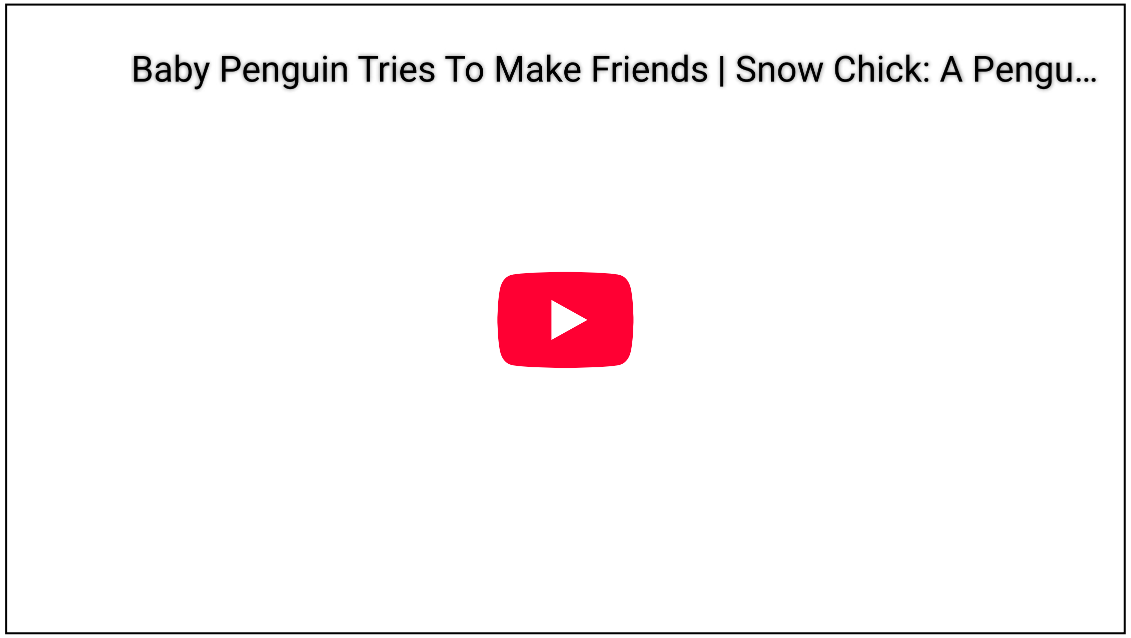


Figure 2: Baby penguin tries to make friends

> Last paragraph in the div block is used as the figure caption.

Finding the figures to include

In places like markdown, YAML, or the command line/shell/terminal, you'll need to use **absolute** or **relative** file paths:

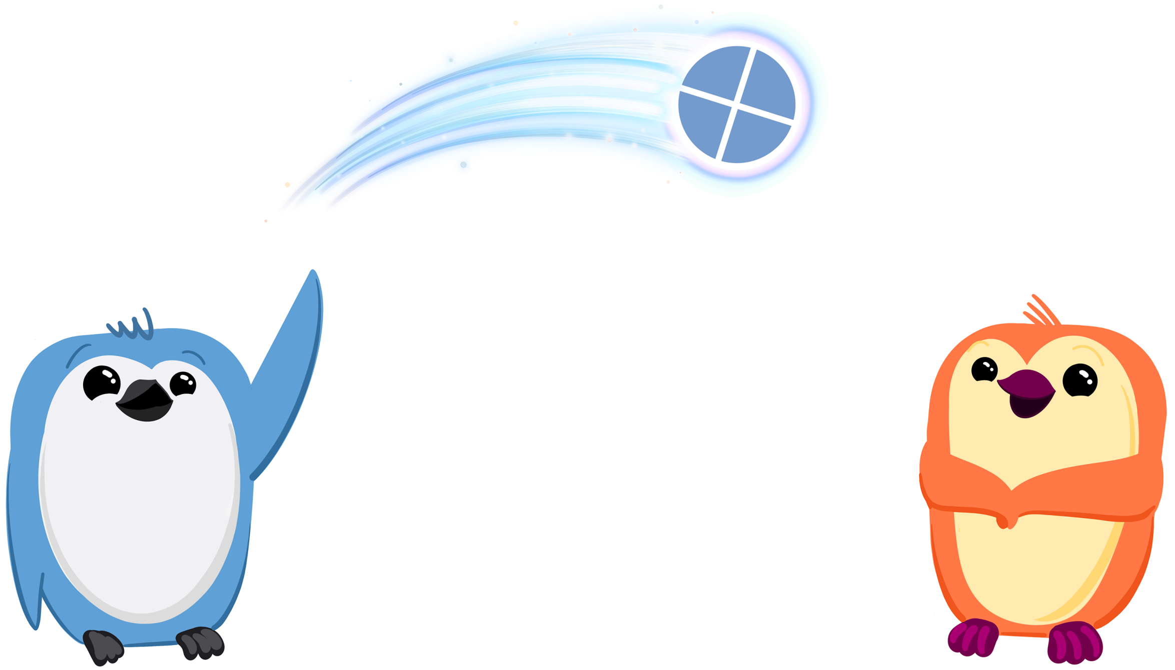
- Absolute = BAD: `"/Users/mine/quarto-monash"` - Whose computer will this work on?

- Relative = BETTER:
  - `"../` = up one directory, `../../` = up two directories, etc.
  - `/..` or `/` = start from `root` directory of your current computer

Figures with code

~~~
```{r}
#| fig-width: 4
#| fig-align: right

knitr::include_graphics("images/penguins-quarto-ball.png")
```
~~~



Referencing paths in R code

Use the **here** package to reference the project root, as `here::here()` always starts at the top-level directory of a `.Rproj`:

```r
here::here()
```

```
[1] "/cloud/project"
```

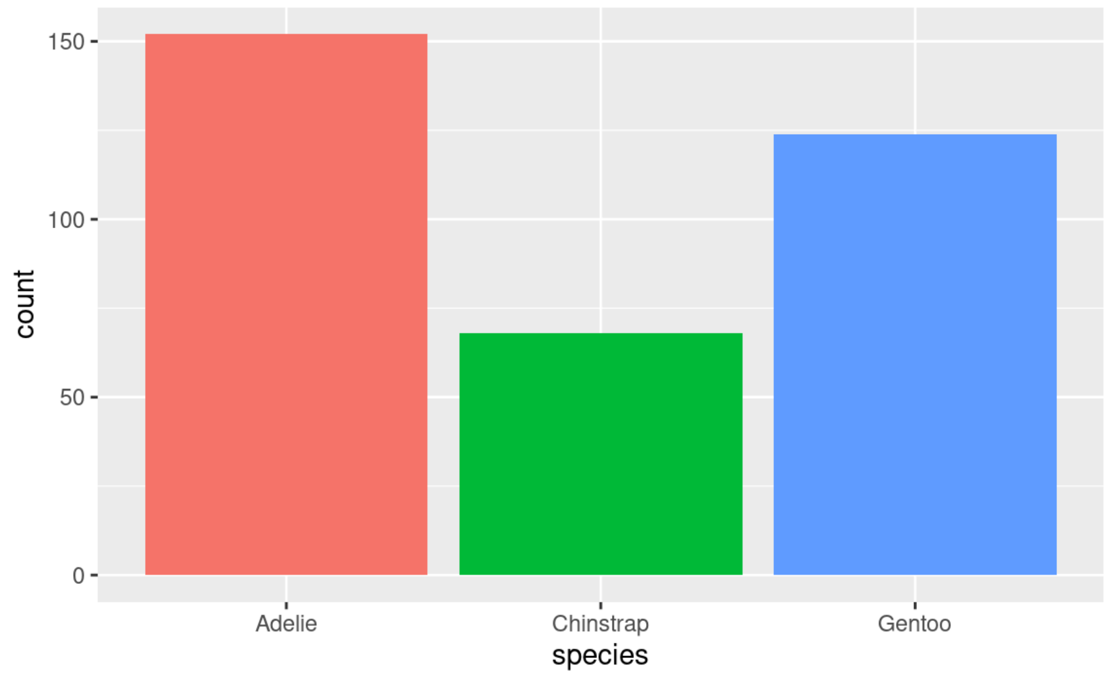
```r
list.files(here::here())
```

```
 [1] "Documents_files"     "Documents.html"     
 [3] "Documents.qmd"       "Documents.rmarkdown"
 [5] "images"              "Lecture_files"      
 [7] "Lecture.html"        "Lecture.pdf"        
 [9] "Lecture.qmd"         "project.Rproj"      
```

Figures from code

~~~
```{r}
#| fig-width: 6
#| fig-asp: 0.618

ggplot(penguins, aes(x = species, fill = species)) +
  geom_bar(show.legend = FALSE)
```
~~~



Cross references

Cross references

- Help readers to navigate your document with numbered references and hyperlinks to entities like figures and tables.
- Cross referencing steps:
  - Add a caption to your figure or table.
  - Give an id to your figure or table, starting with `fig-` or `tbl-`.
  - Refer to it with `@fig-...` or `@tbl-...`.

Figure cross references

The presence of the caption (`Blue penguin`) and label (`#fig-blue-penguin`) make this figure referenceable:

Markdown:

```
See @fig-blue-penguin for a cute blue penguin.
![Blue penguin](images/blue-penguin.png){#fig-blue-pengu
```

Output:

See Figure 3 for a cute blue penguin.

Figure 3: Blue penguin

Table cross references

The presence of the caption (`A few penguins`) and label (`#tbl-penguins`) make this table referenceable:

Markdown:

~~~
See @tbl-penguins for data on a few penguins.

```{r}
#| label: tbl-penguins
#| tbl-cap: A few penguins

head(penguins) |>
  gt()
```
~~~

Output:

See Table 1 for data on a few penguins.

```
head(penguins) |>
  gt()
```

Table 1: A few penguins

species	island	bill_length_mm	bill_depth_mm	flipper_length_mm
Adelie	Torgersen	39.1	18.7	181
Adelie	Torgersen	39.5	17.4	186
Adelie	Torgersen	40.3	18.0	195
Adelie	Torgersen	NA	NA	NA
Adelie	Torgersen	36.7	19.3	193
Adelie	Torgersen	39.3	20.6	190

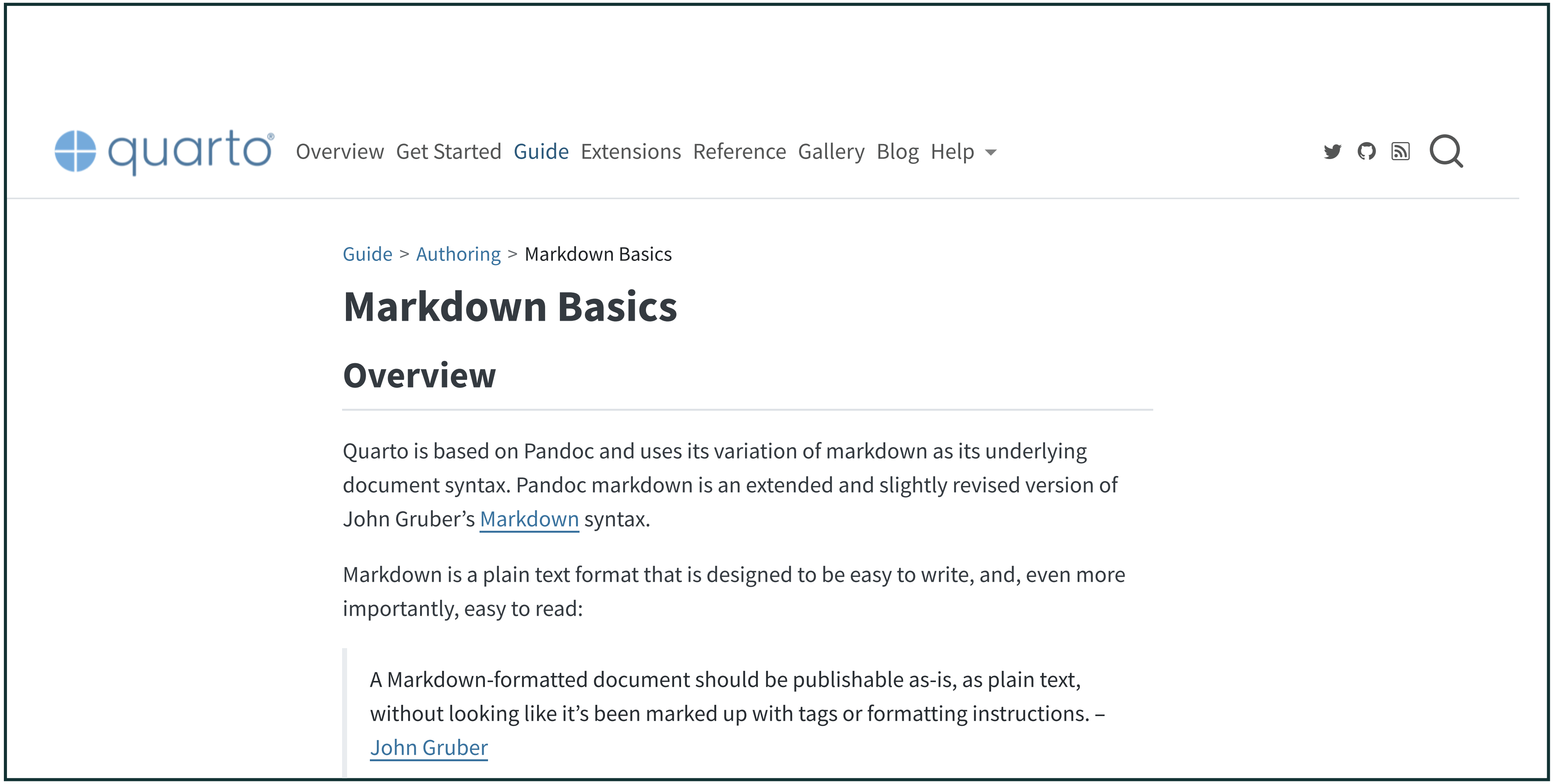
Your turn

- Open `tables-figures.qmd`.
- Follow the instructions in the document, then exchange one new thing you've learned with your neighbor.

10:00

Learn more

quarto.org/docs/authoring/markdown-basics.html



quarto
Overview Get Started Guide Extensions Reference Gallery Blog Help

Guide > Authoring > Markdown Basics

Markdown Basics

Overview

Quarto is based on Pandoc and uses its variation of markdown as its underlying document syntax. Pandoc markdown is an extended and slightly revised version of John Gruber's Markdown syntax.

Markdown is a plain text format that is designed to be easy to write, and, even more importantly, easy to read:

> A Markdown-formatted document should be publishable as-is, as plain text, without looking like it's been marked up with tags or formatting instructions. – John Gruber