

Bausteine Computergestützter Datenanalyse

Leitfaden zur Erstellung von Bausteinen

Maik Poetzsch

2024-04-02

Inhaltsverzeichnis

1	Einleitung	3
2	Installation	3
3	Quarto Markdown Dateien	3
3.1	YAML-Header	3
	Metadaten	4
	Konfiguration	4
	Quellenverwaltung und Zitation	5
3.2	Quarto Markdown	6
	Elementspezifische Optionen	6
	Divs	7
	Programmcode	7
4	Gestaltung von Elementen	8
4.1	Text	8
	Callout Blocks	8
	Querverweise	12
4.2	Grafiken	13
	Grafikoptionen	13
	BCD Nutzung von Vektorgrafiken	15
	Ggf. ergänzen: Flussdiagramme	15
4.3	Videos	15
4.4	Programmcode	16
	Optionen Codeblöcke (bitte ergänzen)	17
	YAML-Header (bitte ergänzen, ggf. oben einfügen)	17
	Optionen für programmierte Grafiken (bitte ergänzen)	17

Mehrsprachige Codebeispiele	17
4.5 Python	17
4.6 R	17
5 Aufbau der Bausteine	18
5.1 Sprache	19
5.2 Barrierefreiheit	19

Lizenzangabe mit maschinenlesbaren Icon nach TULLU(BA)-Regel + Jahr: Titel, Urhebende, Lizenz, Link zur Lizenz. Ursprungsort. (Bearbeitung). (Ausnahmen). Jahr



Bausteine Computergestützter Datenanalyse. Leitfaden zur Erstellung von Bausteinen von Maik Poetzsch ist lizenziert unter [CC BY 4.0](#). Das Werk ist abrufbar unter *<Platzhalter>*. Ausgenommen von der Lizenz sind alle Logos und anders gekennzeichneten Inhalte. 2024

- optional: Zitiervorschlag und BibTeX-Vorlage

1 Einleitung

Die Bausteine Computergestützter Datenanalyse wurden mit [Quarto](#) erstellt. Quarto ist ein quelloffenes Publikationssystem, das die Programmiersprachen Python, R, Julia und Observable sowie verschiedene Publikationsformate wie HTML, PDF, MS Word oder ePub unterstützt. Dieses Dokument ist ein Leitfaden und Gestaltungsrichtlinie zur Bearbeitung und Neuerstellung von Bausteinen.

Nach einer kurzen Einführung zur Installation (Kapitel [2](#)) und allgemeinen Nutzung von Quarto (Kapitel [3](#)), wird die Verwendung von Elementen wie Grafiken und Code-Blöcken in Kapitel [4](#) erläutert. Stilistische Hinweise sind dabei *kursiv* gesetzt.

2 Installation

Um Bausteine zu bearbeiten oder eigene Bausteine im Stil von BCD zu erstellen, benötigen Sie:

- eine lokale Quartodatei,
- eine Entwicklungsumgebung (VS Code, Jupyter, RStudio, Neovim, Text Editor) mit der jeweiligen Quarto-Erweiterung,
- eine Installation der Programmiersprachen Python und R (bzw. der von Ihnen präferierten Programmiersprache) sowie
- die für die jeweilige Programmiersprache verwendeten Pakete.

Siehe: [Quarto Get Started](#)

... Input von Marc Sönnecken

3 Quarto Markdown Dateien

Quarto Markdown Dateien bestehen aus zwei Teilen: Dem YAML-Header und dem in Quarto Markdown geschriebenen Inhalt.

3.1 YAML-Header

YAML (“YAML Ain’t Markup Language”) ist eine Sprache zum Schreiben von Konfigurationsdateien. Der YAML-Header steht am Beginn einer Quarto Markdown Datei. Der YAML-Header enthält die Metadaten eines Dokuments, steuert die technische Ausführung der Dokumentenerstellung und konfiguriert global das Verhalten und Erscheinungsbild von Grafiken, Programmcode und anderen Elementen. Der YAML-Header wird mit `---` begonnen und beendet, Kommentare können mit einer `#` gesetzt werden.


Metadaten

```
---
# Metadaten / meta data
title: "Bausteine Computergestützter Datenanalyse"
subtitle: "Leitfaden zur Erstellung von Bausteinen"
author: "Maik Poetzsch"
date: "2024-03-05" # Jahr-Monat-Tag / year-month-day
---
```

Konfiguration

Für die Dokumentenerstellung können an unterschiedliche Formate angepasste Einstellungen vorgenommen werden. Auf die korrekte Einrückung ist zu achten.

```
format:
  html: # 2 Leerzeichen oder 1 Tab
    option: parameter # 4 Leerzeichen oder 2 Tabs
    option: parameter
  pdf:
    option: parameter
    option: parameter
```

 Hinweis

Ggf. ergänzen: Formatierung mit CSL-Datei

Spracheinstellungen

`lang: de` setzt die Dokumentensprache auf Deutsch. Die Standardeinstellung ist Englisch:
`lang: en`. Weitere Einstellungen können mit `language` auch für verschiedene Sprachen vorgenommen werden. Eine Liste der Optionen findet sich [auf GitHub](#).

```
language:
  de:
    toc-title: Inhalt # Titel des Inhaltsverzeichnisses
  en:
    toc-title: Contents # title for table of contents
```

Hinweis

Hinweis: `lang` ersetzt die einzelne Konfiguration über `crossref`.

```
crossref:
  fig-title: Abbildung
  fig-prefix: Abbildung
  tbl-title: Tabelle
  tbl-prefix: Tabelle
  sec-prefix: Abschnitt
```

Quellenverwaltung und Zitation

Die Quellen werden über eine Bibliografiedatei im Format BibLaTeX (`.bib`) verwaltet. Diese Datei wird im Arbeitsordner angelegt und im YAML-Header mit `'bibliography: bibliography.bib'` eingebunden.

Bibliografiedatei bibliography.bib

In der Bibliografiedatei werden Einträge wie folgt abgelegt:

```
# Printmedien
@book{Hemingway1952,
  title={The Old Man and the Sea},
  author={Hemingway, Ernest},
  year={1952},
  publisher={Charles Scribner's Sons},
  URL={https://www.testurl.com/testurl},
  urldate = {2000-12-31}
}

# Onlineressourcen
@online{Quarto-get-started,
  author = {Quarto},
  title = {Get Started},
  year = {},
  url = {https://quarto.org/docs/get-started/},
  urldate = {2024-02-27}
}

# mehrere Autor:innen
@online{R-Markdown-Cookbook,
```

```

author = {Xie, Yihui and Dervieux, Christophe and Rieder, Emily},
title = {R Markdown Cookbook},
year = {2024},
url = {https://bookdown.org/yihui/rmarkdown-cookbook/},
urldate = {2024-03-04}
}

```

Quarto nutzt Pandoc zur Formatierung von Zitaten und Quellennachweisen. Pandoc nutzt standardmäßig den [Chicago-Stil](#), das Nachweise im Nummern- und im Autor-Jahr-System definiert. *In den Bausteinen werden Quellen im Autor-Jahr-System CMOS nachgewiesen.*

Ziterstil *Autor-Jahr-System* `biblio-style: authoryear` [CMOS Kurzanleitung](#)

i Hinweis

Hinweis: Das Erscheinungsbild des Quellenverzeichnisses unterscheidet sich in HTML und PDF leicht.

Ergänzen: bausteinübergreifende Quellenverwaltung.

Marc: Wenn man ein Quarto Projekt anlegt, kann man global den Pfad setzen.

3.2 Quarto Markdown

Quarto Markdown ist eine Erweiterung von Markdown, einer maschinenlesbaren Auszeichnungssprache für die Formatierung von Texten und weiteren Elementen wie Grafiken oder Programmcode. Eine Übersicht über die von Quarto Markdown unterstützten Formate bieten die [Quarto Hilfeseiten](#). Quarto Markdown basiert auf Pandoc. Das [Pandoc Handbuch](#) kann bei spezifischen Fragen oder Problemen weiterhelfen.

Elementspezifische Optionen

Das Verhalten und Erscheinungsbild einzelner Elemente kann abweichend von den globalen Einstellungen durch elementspezifische Optionen kontrolliert werden. Abhängig vom jeweiligen Element werden Optionen mit einer bestimmten Syntax übergeben:

- Codezellen werden durch führende Kommentarzeilen parametrisiert. (Anders als in R Markdown sollen Zelloptionen nicht in der geschweiften Klammer übergeben werden.)
 - In Python, R und Julia mit `#| option: parameter`
 - In Observable JavaScript mit `//| option: parameter`

- Objekte wie Überschriften, Callout Blocks, Divs, Grafiken und Tabellen werden mit geschweiften Klammern gesteuert `{option="parameter"}`

Die Konfigurationsmöglichkeiten werden in Kapitel 4 erläutert.

Divs

Divs bieten vielfältige Möglichkeiten, Abschnitte zu formatieren. Divs werden mit mindestens drei Doppelpunkten `:::` eingeleitet und beendet (vier und mehr Doppelpunkte helfen bei verschachtelten Divs, den Überblick zu behalten). Optionen werden in geschweiften Klammern übergeben. Siehe: [Quarto Divs](#)

Besondere Bedeutung haben Divs für:

- das Layout in mehreren Zeilen oder Spalten und die Abstandsformatierung (siehe Lizenzhinweis am Anfang des Dokuments). [Quarto Figure Panels](#)
- Layout von Code-Blöcken in einem [Tabset Panel](#)
- Conditional Content zur formatabhängigen Einbindung von Inhalten. [Quarto Conditional Content](#)
- Erweiterte Möglichkeiten für Querverweise. [Quarto Cross-Reference Div Syntax](#)
- Sonderformate wie Callout Blocks (siehe Kapitel 4).

Programmcode

Quarto Markdown kann Code von verschiedenen Programmiersprachen ausführen: Python, R, Julia und Observable JavaScript. Dazu unterstützt Quarto die Engines Knitr und Jupyter zur dynamischen Berichterstellung. [Quelle: Quarto Frequently Asked Questions](#)

Python-Code wird mit Jupyter verarbeitet. Dazu muss eine lokale Installation von Python vorhanden sein. Die Installationsdatei sollte von der [Python Homepage](#) bezogen werden.

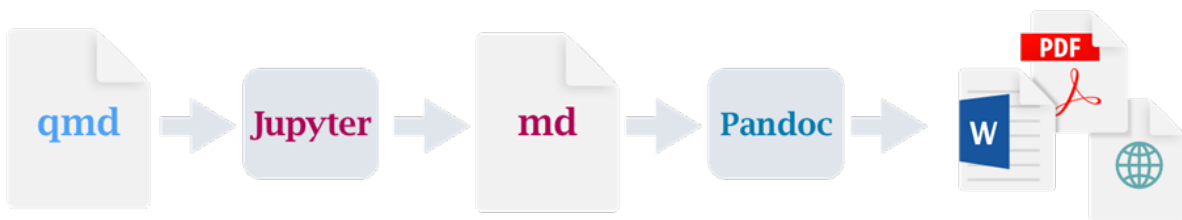


Abbildung 1: Jupyter Engine, [Quelle](#)

R-Code wird mit Knitr verarbeitet. Dazu muss eine lokale Installation von R vorhanden sein, in der die Pakete `knitr`, `rmarkdown` sowie für die Ausführung von Python-Code das Paket `reticulate` installiert sind.



Abbildung 2: Knitr Engine, [Quelle](#)

4 Gestaltung von Elementen

i Hinweis

Ggf ergänzen: Verwendung einer mystyle.css

4.1 Text

Regulärer Text wird in [Markdownsyntax](#) durch Sonderzeichen formatiert. Diese Sonderzeichen können durch ein vorangestellte Backslash \ in der Ausgabe sichtbar gemacht werden.

Callout Blocks






Callout Blocks eignen sich insbesondere dazu, textuelle Inhalte hervorzuheben. Callout Blocks können umfangreich angepasst werden und Elemente wie Grafiken oder Code-Blöcke enthalten. Die Typen werden mit der Spracheinstellung im YAML-Header `lang: de` lokalisiert.


```
:::{.callout-note}
```

Es gibt fünf Typen von callouts:


```
`note`, `tip`, `warning`, `caution`, und `important`.
```

```
:::
```


 Hinweis Es gibt fünf Typen von callouts: <code>note</code> , <code>tip</code> , <code>warning</code> , <code>caution</code> , und <code>important</code> .	 Tipp Es gibt fünf Typen von callouts: <code>note</code> , <code>tip</code> , <code>warning</code> , <code>caution</code> , und <code>important</code> .
 Warnung Es gibt fünf Typen von callouts: <code>note</code> , <code>tip</code> , <code>warning</code> , <code>caution</code> , und <code>important</code> .	 Vorsicht Es gibt fünf Typen von callouts: <code>note</code> , <code>tip</code> , <code>warning</code> , <code>caution</code> , und <code>important</code> .
 Wichtig Es gibt fünf Typen von callouts: <code>note</code> , <code>tip</code> , <code>warning</code> , <code>caution</code> , und <code>important</code> .	

 Auklappbarer Callout Block

`collapse="true"` macht den Tipp in HTML aufklappbar, z. B. um Tipps zu Aufgaben zu geben.

 Callouts können auch verschachtelt werden

Die erste Überschrift im Markdownformat wird als Titel benutzt, wenn keiner spezifiziert wurde.

Das Symbol kann unterdrückt werden


```
icon="false"
```

Siehe dazu: [Quarto Callout Blocks](#)

Definieren von eigenen Callout Umgebungen

Mit folgendem Vorgehen lassen sich eigene Callout-Umgebungen definieren wobei diese zum jetzigen Stand (April 2024) in der PDF farblich nicht anpassbar sind.

Zu erst wird eine Ordnerstruktur für Quarto-Erweiterungen benötigt:

```
_extension
- _extension.yml
- callout_definition.lua
```

```

- theme.scss
_quart.yml
mein_dokument.qmd

```

Die `*_extension.yml*` listet dabei alle benutzen Filter auf, in diesem Fall `callout_definition.lua`. Hier lässt sich auch eine `.scss` Datei unterbringen, in der die Änderungen am Aussehen der HTML gespeichert sind. Diese werden dann in der Haupt yml Datei mit `courseformat-html` aufgerufen.

```

title: Course Page Format
author: Marc Fehr
version: 1.0.0
contributes:
  formats:
    html:
      theme: [default, theme.scss]
  filters:
    - callout_definition.lua

```

Die Definition des Filters und damit der eigenen Callout Umgebung sieht folgendermaßen aus:

```

function Div(div)
  -- process exercise
  if div.classes:includes("callout-definition") then
    -- default title
    local title = "Definition"
    -- Use first element of div as title if this is a header
    if div.content[1] ~= nil and div.content[1].t == "Header" then
      title = pandoc.utils.stringify(div.content[1])
      div.content:remove(1)
    end
    -- return a callout instead of the Div
    return quarto.Callout({
      type = "definition",
      content = { pandoc.Div(div) },
      title = title,
      collapse = false
    })
  end
end
end

```

In der *theme.scss* wird dann die Darstellung der callout-Umgebung festgelegt. Da es sich hier um eine reine HTML Anpassung handelt findet sich das Resultat nur in HTML-Ausgabe und nicht in der PDF. Über die Hexadezimalwerte können dabei die Farben gesteuert werden. In der PDF erscheint ein solcher callout-Block dann in der Farbe Hellgrau. Im letzten (auskommentierten) Abschnitt lässt sich ein Symbol (beispielsweise ein Ausrufezeichen für eine Warnung-Umgebung) festlegen. Dabei wird auf die *Font Awesome*-Bibliothek zurück gegriffen.

```
/*-- Importing fa icons --*/
@import url("https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0/css/all.min.css");

/*-- scss:rules --*/

// Exercise callout styling
div.callout-definition.callout {
  border-left-color: #aea545;
}

div.callout-definition.callout-style-default > .callout-header {
  background-color: #474765;
}

/* Hier kann man ein Icon hinzufügen
.callout-definition > .callout-header::before {
  font-family: "Font Awesome 5 Free";
  content: "\f303";
  margin-right: 10px;
}
*/
```

Verwendungsvorschlag für die Bausteine

Note 1: Beispiel Note

callout-note für Beispiele

Querverweis mit **#nte-ID** **Nicht von lang: de, keine crossref-Option betroffen**

Definition Begriff XY

callout-important für Definitionen - der definierte Begriff wird als Überschrift verwendet. **Ggf. grau**

Querverweis mit `#imp-ID`

💡 Tipp

`callout-tip` aufklappbar für Lösungshilfen und Lösungen

Querverweis mit `#tip-ID`

⚠️ Hinweis

`callout-warning appearance="simple"` für Hinweise

Querverweis mit `#wrn-ID`

Achtung Der Querverweis auf einen Callout Block bleibt Englisch: Note [1](#). Dass kann man aber notfalls manuell lösen: Querverweis auf Hinweis [1](#):

Querverweise

Um Querverweise zu setzen, muss das Zielement mit einer ID versehen werden. Elementen (z. B. Code-Blöcke, Grafiken, Überschriften) wird eine ID in geschweiften Klammern übergeben, die ID muss innerhalb der Klammer an erster Stelle stehen. Ein Beispiel, in dem für die eingebundene Grafik eine ID vergeben sowie die Größe der Grafik mit der Option `width` eingestellt wird:

```
![] (grafiken/working_code_CC0){#fig-programmieren width="33%"}
```



Abbildung 3

Mit der ID kann ein Querverweis auf die Abbildung gesetzt werden: `@fig-programmieren` erzeugt den Querverweis [Abbildung 3](#). Der Verweistext kann angepasst werden: `[Grafik @fig-programmieren]` erzeugt den Querverweis [Grafik 3](#).

Sowohl die ID der Grafik als auch der Querverweis beinhalten ein Präfix, das den Typ des Elements ausweist.

Reservierte Präfixe

In Quarto sind verschiedene Präfixe für die Erstellung von Querverweisen reserviert: fig, tbl, lst, tip, nte, wrn, imp, cau, thm, lem, cor, prp, cnj, def, exm, exr, sol, rem, eq, sec.

Siehe dazu: [Quarto Cross References](#)

4.2 Grafiken

Grafiken können lokal oder aus dem Internet eingebunden werden. Der lokale Dateipfad wird ausgehend vom aktuellen Arbeitsverzeichnis angegeben. *Grafiken werden im Unterordner “grafiken” abgelegt.*

Speicherort Grafiken *Unterordner im Arbeitsverzeichnis “grafiken”*

Die Syntax lautet:

```
![Grafiküberschrift] (grafiken/Dateiname_Dateiname Dateiname.png)
![Grafiküberschrift] (https://beispiellink.de/einbild.png)
```

Arbeitsverzeichnis ermitteln

Das aktuelle Arbeitsverzeichnis kann mit einem Codeblock mit dem entsprechenden Befehl angezeigt werden (hier ohne Ausgabe):

In R:

```
print(getwd())
```

In Python:

```
import os
print(os.getcwd())
```

Grafikoptionen

Grafikoptionen können global im YAML-Header definiert werden.

```

---
cap-location: top
fig-align: center
---
```

Hinweis



Eine Grafik ohne Beschriftung oder `#fig-ID` folgt der globalen Einstellung nicht. Das obenstehende Bild steht in HTML linksbündig, obwohl im YAML-Header `fig-align: center` konfiguriert ist. Quarto unterscheidet intern zwischen unbeschrifteten Bildern (`image`) und beschrifteten Grafiken (`figure`). Siehe <https://github.com/quarto-dev/quarto-cli/issues/6509#issuecomment-1677657369>.

Grafikoptionen können auch elementweise gesetzt werden. Optionen für einzelne Grafiken werden in geschweiften Klammern übergeben und mehrere Optionen durch Leerzeichen voneinander getrennt. Im folgenden Beispiel wird eine `#fig-ID` vergeben, ein Alternativtext mit `fig-alt=""` angelegt, die Ausrichtung mit `fig-align=""` linksbündig sowie die Größe der Grafik mit `width=""` eingestellt. `width` stellt die Breite der Grafik ein, die Höhe wird automatisch berechnet.

```

![Grafik mit elementweiser Option "left">(grafiken/working_code_CC0.png){#fig-Grafik-mit-Opt.
fig-alt="Eine Person programmiert am Computer" fig-align="left" width="33%"}

```



Abbildung 4: Grafik mit elementweiser Option “left”

Grafiken *Globale Einstellungen zentriert* `fig-align: center`

Beschriftung unterhalb `cap-location: bottom` (default)

Einbindung mit `#fig-ID` und Alternativtext `fig-alt="Alternativtext"` (ausgenommen dekorative Grafiken)

BCD Nutzung von Vektorgrafiken

Die Bausteine sind für den Export in HTML und PDF konzipiert. Der Export von Grafiken nach PDF erfolgt über LaTeX und eine PDF-Renderengine. Somit werden nur Formate unterstützt, die in LaTeX und in PDF unterstützt werden. Dies betrifft insbesondere

- Vektorgrafiken: Um Vektorgrafiken im Format SVG zu verarbeiten, wird die Bibliothek `Librsvg` benötigt. Siehe: [Quarto: PDF Format Improvements](#)
- GIF: Das PDF-Format unterstützt animierte Bilddateien nicht bzw. nur in bestimmten Kombinationen aus Renderengine und PDF Reader. Quartos Standardengine TinyTeX unterstützt animierte Bilddateien nicht.

Um unabhängig vom Exportformat SVG-Dateien zu nutzen, werden diese ohne Formatendung eingebunden.

`![Grafiküberschrift] (Unterordner/Dateiname_Dateiname Dateiname)`

Das im jeweiligen Exportformat verwendete Grafikformat wird im YAML-Header definiert.

```
format:
  html:
    default-image-extension: svg
  pdf:
    cite-method: biblatex
    default-image-extension: pdf
```

Hinweis

Auf GitHub wandelt ein Skript alle SVG-Grafiken automatisch in PDF um.

Ggf. ergänzen: Flussdiagramme

Flussdiagramme siehe <https://quarto.org/docs/authoring/markdown-basics.html#diagrams>

4.3 Videos

Speicherort von Videos *Unterordner im Arbeitsverzeichnis “videos”*

Die Syntax zur Einbindung von Videos lautet:

```
# Allgemein
{{< video Dateipfad >}}

# Beispiel
{{< video videos/hermione-slow-clap.mp4 >}}
```

[Videos/hermione-slow-clap.mp4](#)

(lizenzfreies Beispiel einfügen)

4.4 Programmcode

Code wird mit folgender Syntax eingebunden:

```
```{python}
print("Hallo Welt")
```
```

Hallo Welt

Dabei können verschiedene *Flags* gesetzt werden. Diese Flags können entweder lokal in der Programmierungsumgebung oder aber global in der yml-Datei gesetzt werden.

Mit `#!/ echo: false` (standartmäßig **true**) kann die Ausgabe des Codes unterdrückt werden. Es wird im Ausgabedokument dann auch nur die Ausgabe des Codes dargestellt.

```
```{python}
#!/ echo: false
print("Hallo Welt")
```
```

Hallo Welt

Im Gegensatz dazu bestimmt `#!/ output: false` (standartmäßig **true**) ob die Ausgabe des Codes dargestellt werden soll. Mit `#!/ output: asis` wird der Output als runformatiert als reine Textausgabe dargestellt:

```
```{python}
#!/ output: false
print("Hallo Welt")
```
```


Optionen Codeblöcke (bitte ergänzen)

```
#| include: true # = output + echo
#| label: load-packages
#| results: hold # process code first, then print output
```

Ergänzen: Optionen Im YAML-Header, Elementspezifische Optionen, Optionen für programmierte Grafiken

YAML-Header (bitte ergänzen, ggf. oben einfügen)

```
format:
  html:
    code-copy: true
```

Optionen für programmierte Grafiken (bitte ergänzen)

```
#| fig-cap: "Beschriftung"
#| label: fig-ID
#| fig-alt: "Alternativtext"
```

Mehrsprachige Codebeispiele

Für die Präsentation von Programmcode in mehreren Sprachen können [Tabset Panel](#) genutzt werden. Diese werden mit einer Div {.panel-tabset} gesetzt, innerhalb derer neue Tabs durch eine Überschrift hinzugefügt werden ## Python.

4.5 Python

```
import os
print(os.getcwd())
```

4.6 R

```
print(getwd())
```

Zusammenspiel von Python und R in Quarto

Wird ein Block mit R-Code in ein Dokument eingefügt, wird dieses mit Knitr gerendert. Python-Code wird über das Paket ‘reticulate’ mit Knitr verarbeitet.

5 Aufbau der Bausteine

Die Bausteine folgen einem einheitlichen Aufbau:

- Voraussetzungen
 - Inhaltliche Voraussetzungen
 - vorher zu bearbeitende Bausteine
 - verwendete Pakete und Datensätze / -quellen
 - geschätzte Bearbeitungszeit
- Lernziele (Wissen, Kompetenzen, Leitfragen)
- Inhalt (gerne abwechslungsreich gestalten)
 - Theorie
 - Beispiele
 - Übungen
- Das Wichtigste (vielleicht als Video)
- Lernzielkontrolle
 - Kompetenzquiz (ggf. aufklappbarer Callout Block, Textverweis für PDF, polierte Lösungen evntl. via Lumi später entscheiden)
 - Übungsaufgaben (kleine Projekte)
- Prüfungsaufgaben (ohne Lösungen)

5.1 Sprache

Die Ansprache der Leser:innen erfolgt in der Höflichkeitsform “Sie / Ihr”. Gegendert wird mit dem Doppelpunkt: Er:Sie ist Busfahrer:in. Es werden nur Personen gegendert. Nicht gegendert wird zum Beispiel: “die Anbieter von Plagiatserkennungssoftware”. Die Anbieter sind Unternehmen.

Eine Alternative zum Gendern mit Doppelpunkt ist die Verlaufsform, z. B. “Studierende”.

5.2 Barrierefreiheit

Siehe [Checklisten Barrierefreiheit](#) Checklisten: Barrierefreiheit in der digitalen Lehre. Hochschuldidaktik im digitalen Zeitalter.nrw; Kompetenzzentrum digitale Barrierefreiheit.nrw. CC BY 4.0.