typstage

Version 0.1.0

Animated HTML talks and a PDF handout from one source.

`typstage` setzt Präsentationen mit Typst und bewegt sie im Browser. Jede Folie wird als SVG gesetzt — die Anordnung ist deshalb dieselbe wie im PDF —, und was sich bewegen soll, wird angemeldet: Einblendungen, Magic Move, Folienübergänge, Medien. Aus derselben Quelle entstehen eine animierte HTML-Präsentation, ein Foliensatz als PDF und ein Handout zum Mitschreiben.

AUTOREN Alexander Schulz <@Loewe1000>
LIZENZ MIT
VERWEISE GitHub · Beispiele · English

Inhalt

1 Über dieses Paket

Das `typstage`-Paket erzeugt aus einer einzigen Typst-Datei eine animierte Präsentation für den Browser – und aus derselben Quelle eine PDF. Der Satz dahinter lautet: **Typst setzt, der Browser bewegt.** Typst setzt jede Folie als SVG und schreibt sie so in die HTML-Datei; im Browser steht sie deshalb genau wie auf dem Papier. Was sich bewegen soll, meldet man im Quelltext an. Eine Folie bleibt dabei eine Folie: die PDF hat eine Seite je Folie, nicht eine je Schritt.

1.1 Fünf Wörter, die dieses Handbuch benutzt

Folie — Ein Bild, von Typst einmal gesetzt. Eine Seite der PDF, ein `.ts-slide` in der HTML.

Schritt — Ein Druck auf die Pfeiltaste. Eine Folie kann mehrere davon halten; am Ende führt der nächste Druck zur nächsten Folie. Die Adresszeile zählt Schritte, die Fußzeile zählt Folien.

Element — Ein Stück Folie, das die Laufzeit anfassen darf. `anim`, `stagger`, `alternatives`, `morph`, `embed`, `video` und `flipbook` erzeugen je eines.

Morph — Dasselbe benannte Element zweimal – auf zwei Folien oder auf zwei Schritten einer Folie. Dazwischen fliegt es, Zeichen für Zeichen.

Sprecheransicht — Dieselbe Datei ein zweites Mal geöffnet, mit `#speaker` in der Adresse. Sie trägt Notiz, Uhr und den nächsten Schritt.

1.2 Wo es zwischen den anderen steht

`touying` und `polylux` sind ausgereift, haben weit mehr Themes und machen PDF. Wer einen gewöhnlichen PDF-Vortrag will, nimmt eines davon. `reveal.js`, `Slidev` und `Quarto` animieren im Browser, aber der Satz ist dort HTMLs, nicht Typsts.

Am nächsten kommen die Typst-Pakete, die selbst HTML schreiben. `touying-exporter` setzt je Folie ein SVG und packt die Folge mit `impress.js` in eine Datei. `slipst` folgt slipshow und gibt die feste Folie ganz auf: dort scrollen „slips“ von oben nach unten. Auf der PDF-Seite lohnen daneben `mosaic`, das die Folien aus denselben Überschriften schneidet wie dieses Paket (`=` für den Abschnitt, `==` für die Folie), und `slydekit`. Drei der siebzehn Beispieldecks sind Adaptionen von mosaic-Decks, damit dieser Vergleich auf dem Bildschirm steht und nicht in der Prosa.

Was dieses Paket kann und keines davon: Ein benanntes Stück steht auf Folie n an einer Stelle und auf Folie n+1 woanders. Es fliegt dorthin, im besten Fall Zeichen für Zeichen, sodass sich eine Gleichung sichtbar selbst umschreibt. Möglich ist das, weil hier je *Zustand* ein SVG entsteht und nicht je Folie.

> **ACHTUNG**
> Der Preis, damit er vor der ersten Zeile Code steht: Die Folien sind SVG-Umrisse. Nichts im Browser ist auswählbar oder durchsuchbar, und ein Bildschirmleser sieht überhaupt nichts. Das Kapitel „Weitergeben“ sagt, was sich dagegen tun lässt.

Dieses Handbuch ist nach Vorhaben geordnet, nicht nach Funktionen:

1. **Die erste Präsentation** – von der leeren Datei zur laufenden HTML
2. **Ein Deck von Anfang bis Ende** – ein Vortrag, zu Ende gebaut
3. **Eine Folie Schritt für Schritt aufdecken** – `pause`, `stagger`, `anim` und der Schrittzeiger
4. **Etwas vorführen statt behaupten** – Applet, Video, Daumenkino
5. **GeoGebra** – Konstruktionen, die den Schritten der Folie folgen
6. **Desmos** – derselbe Weg, ein anderer Rechner
7. **Eine Rechnung entwickeln** – Magic Move über mehrere Folien

8. **Den Vortrag halten** – Tasten, Sprecheransicht, Zeichnen, die zwei Uhren
9. **Aus einer Quelle drei Ausgaben** – Präsentation, Foliensatz, Handout
10. **Das eigene Aussehen** – Themes, Farben, Leinwand, Bausteine
11. **Weitergeben** – wo die Datei liegt und wie groß sie wird
12. **Was es nicht kann** – Barrierefreiheit, Reichweite, die harten Grenzen
13. **Wenn nichts passiert** – die Stolpersteine, in der Reihenfolge, in der man über sie fällt
14. **API-Referenz** – vollständige Funktionsdokumentation

> **HINWEIS**
> Die gesetzten Beispiele dieses Handbuchs sind Papier und zeigen deshalb den Endzustand – alles auf einmal. Was im Browser nacheinander geschieht, steht im Text daneben.

2 Die erste Präsentation

Ziel dieses Kapitels: eine vollständige, vorführbare Präsentation, ohne Umwege.

2.1 Eine Datei genügt

Mehr braucht es nicht als Import, Show-Regel und Überschriften. Diese Datei ist vollständig und lässt sich abtippen:

```
#import "@preview/typstage:0.1.0": *

#show: presentation.with(
  title: [Der Satz des Pythagoras],
  subtitle: [Eine Herleitung in vier Schritten],
  author: [Mathematik · Klasse 9],
  date: datetime.today(),
  transition: "slide",
)

= Worum es geht

== Die Behauptung

#speaker-note[Erst die Zerlegung zeigen, dann die Formel -- nicht umgekehrt.]

#stagger[
  - Ein rechtwinkliges Dreieck hat zwei Katheten und eine Hypotenuse.
  - Über jeder Seite steht ein Quadrat.
  - Die beiden kleinen sind zusammen so groß wie das große.
]

#v(1em)

#anim(callout[Genau das behauptet der Satz.], enter: "scale")

== Und das ist die Formel

#statement[$ a^2 + b^2 = c^2 $]
```

Daraus entstehen vier Folien: die Titelfolie aus `title`, eine Abschnittsfolie aus `=`, und je eine gewöhnliche Folie aus den beiden `==`. Der Text bis zur nächsten Überschrift ist der Rumpf einer Folie.

So sieht der Rumpf der Folie „Die Behauptung“ aus, wenn alle Schritte abgelaufen sind:

```
#stagger[
  - Ein rechtwinkliges Dreieck hat zwei Katheten und eine Hypotenuse.
  - Über jeder Seite steht ein Quadrat.
  - Die beiden kleinen sind zusammen so groß wie das große.
]

#v(1em)

#anim(callout[Genau das behauptet der Satz.], enter: "scale")
```

- Ein rechtwinkliges Dreieck hat zwei Katheten und eine Hypotenuse.
- Über jeder Seite steht ein Quadrat.
- Die beiden kleinen sind zusammen so groß wie das große.

MERKE
Genau das behauptet der Satz.

Im Browser erscheinen die drei Punkte nacheinander, der Merksatz auf dem vierten Tastendruck.

2.2 Zweimal übersetzen

Dieselbe Datei ergibt zwei Ausgaben, ohne Nachbearbeitung dazwischen.

```
typst compile vortrag.typ vortrag.html --format html --features html
typst compile vortrag.typ vortrag.pdf
```

Die erste Zeile ergibt die animierte Präsentation als eine einzige Datei, die ein Doppelklick öffnet – ohne Server und ohne Netz. Die zweite ergibt den Foliensatz zum Ausdrucken.

HINWEIS
Der HTML-Export ist in Typst 0.15 Vorschau und verlangt deshalb `--features html`. Die Warnung, die Typst dabei ausgibt, betrifft den Export im Allgemeinen, nicht dieses Paket.

ACHTUNG
Inhalt vor der ersten Überschrift gehört zu keiner Folie: Text bricht das Übersetzen ab, ein Bild verschwindet wortlos. Und ein `#set heading` nach der Show-Regel erreicht die Folientitel nicht mehr – dafür gibt es `style`, siehe „Das eigene Aussehen„.

2.3 Mehr als zwei Ebenen

Die Vorgabe schneidet das Deck an der zweiten Ebene: `=` wird eine Abschnittsfolie, `==` eine Folie. `slide-level` verschiebt diesen Schnitt. Eine Überschrift **über** der Ebene wird zur Abschnittsfolie, eine Überschrift auf ihr oder darunter wird zur Folie.

```
#show: presentation.with(title: [Analysis I], slide-level: 3)
= Teil I -- Grenzwerte
== Folgen
=== Was eine Folge ist
Eine Abbildung von den natürlichen in die reellen Zahlen.
=== Konvergenz
Für jedes Epsilon gibt es ein N.
== Reihen
=== Partialsummen
Die Summe der ersten n Glieder.
```

Aus `= Teil I` und `== Folgen` werden Abschnittsfolien, aus jedem `===` eine Folie. Die Trennfolien für **beide** Ebenen fallen von selbst an. `slide-level: 1` macht jede Überschrift zu einer Folie; das Deck hat dann keine Struktur-Ebene mehr.

Die fünf mitgelieferten Themes zeichnen eine tiefere Ebene ruhiger: der Titel wird kleiner, und darüber steht, worunter der Abschnitt hängt. Ein eigenes Theme liest dafür `s.depth` und `s.parents` und darf beide übergehen.

Was das Deck über seine Gliederung weiß, steht in `info()`: `section` ist die Ebene direkt über der Folie, `levels` hat einen Eintrag je Struktur-Ebene, und `outline` ist die ganze Gliederung.

2.3.1 Text, der zu keiner Folie gehört

Eine Abschnittsfolie ist ein ganzes Bild, das das Theme zeichnet; sie hat keinen Rumpf. Text zwischen ihrer Überschrift und der nächsten bricht das Übersetzen deshalb ab. Ein Satz zwischen `= Der Beweis` und `== Die Zerlegung` meldet `content between the heading "Der Beweis" and the next one belongs to no slide`:

```
#show: presentation.with()
= Der Beweis
Dieser Satz gehört zu keiner Folie und hält das Übersetzen an.
== Die Zerlegung
```

Er gehört unter die Folienüberschrift:

```
#show: presentation.with()
= Der Beweis
== Die Zerlegung
Dieser Satz gehört zur Folie und wird gesetzt.
```

Zweierlei schränkt die Prüfung ein: Text **vor** der ersten Überschrift wird nur abgewiesen, wenn das Deck überhaupt eine Überschrift hat. Und geprüft wird nur in der Überschriften-Schreibweise; wer Folien als Argumente übergibt, schreibt jeden Rumpf ohnehin selbst hin.

2.4 Wenn die Folien berechnet werden

Auch Überschriften, die erst beim Setzen entstehen, werden zu Folien. Eine Schleife über eine Liste ergibt so eine Folie je Eintrag:

```
#for stoff in ("Wasser", "Luft", "Erde") [
  == #stoff
  Etwas über #stoff.
]
```

Wo die Folien vollständig aus Daten entstehen, lassen sie sich auch einzeln übergeben. Dann ist jede Folie ein Funktionsaufruf, und eine Liste von Folien lässt sich mit `..` weiterreichen wie jedes andere Array:

```
#presentation(
  title-slide(title: [Der Satz des Pythagoras], author: [A. Schulz]),
  section[Der Beweis],
  slide([Die Zerlegung], note: [Zuerst das Quadrat zeigen.])[
    Der Text der Folie.
  ],
)
```

Beide Schreibweisen ergeben dieselbe Ausgabe; `presentation` erkennt an dem, was es bekommt, welche gemeint ist. Die Form mit Überschriften ist der Normalfall.

3 Ein Deck von Anfang bis Ende

Die übrigen Kapitel zeigen je ein Mittel. Dieses zeigt einen ganzen Vortrag: eine einzige Datei, von der leeren Zeile bis zum Handout, und jeder Schritt fügt genau eine Sache hinzu.

Der Stoff ist eine Aufgabe aus der Mittelstufe: **Wie hoch ist der Turm?** Ein Stab von 1,20 m wirft 0,90 m Schatten, der Turm wirft 21 m. Gesucht ist seine Höhe.

3.1 Die leere Datei

Zwei Zeilen sind ein Deck. Die erste holt das Paket, die zweite sagt, dass dieses Dokument eine Präsentation ist.

```
#import "@preview/typstage:0.1.0": *
#show: presentation.with(title: [Wie hoch ist der Turm?])
```

Übersetzt wird das zweimal, aus derselben Datei:

```
typst compile turm.typ turm.html --format html --features html
typst compile turm.typ turm.pdf
```

Die HTML öffnet man im Browser und blättert mit den Pfeiltasten. Bis hierhin gibt es nur die Titelfolie – `title:` genügt, damit sie entsteht.

3.2 Die erste Folie

Eine Überschrift der zweiten Ebene ist eine Folie, der Text darunter ihr Rumpf. Eine Überschrift der ersten Ebene ist eine Abschnittsfolie.

```
= Die Frage

== Ein Stab und ein Turm

Ein Stab von 1,20 m wirft einen Schatten von 0,90 m.
Der Turm wirft 21 m. Wie hoch ist er?
```

Mehr Struktur braucht es nicht.

3.3 Was nacheinander erscheinen soll

Drei Beobachtungen, eine nach der anderen. `stagger` zerlegt eine Aufzählung an ihren Punkten und gibt jedem einen eigenen Schritt.

```
== Was wir sehen

#stagger[
  - Die Sonne steht für beide gleich hoch.
  - Also ist der Winkel derselbe.
  - Also sind die Dreiecke ähnlich.
]
```

Jetzt hat die Folie vier Schritte: den Rumpf und drei Punkte.

3.4 Ein Kasten, der hängen bleibt

Der Satz, auf den es ankommt, gehört nicht in die Aufzählung. `callout` setzt ihn mit einem Balken an der linken Seite ab.

```
== Was wir sehen

#stagger[
  - Die Sonne steht für beide gleich hoch.
  - Also ist der Winkel derselbe.
  - Also sind die Dreiecke ähnlich.
]

#callout[
  In ähnlichen Dreiecken stehen entsprechende Seiten im selben Verhältnis.
]
```

Die Überschrift des Kastens folgt der Sprache des Dokuments – auf Deutsch „Merke“. `title:` ändert sie, `title: none` lässt sie weg.

3.5 Die Formel, die sich selbst umschreibt

Dieselbe Formel steht auf zwei Folien an zwei Stellen und fliegt dazwischen, Zeichen für Zeichen. Dafür ist nur eines zu tun: beide Folien nennen sie beim selben Namen, und der Name ist ein Label.

```
== Das Verhältnis

#align(center, morph(<turm>, $ h / 21 = 1.2 / 0.9 $))

== Nach h aufgelöst

#align(center, morph(<turm>, $ h = 21 dot 1.2 / 0.9 $))
```

Im Browser wandern `h`, der Bruchstrich und die Zahlen an ihre neue Stelle, statt zu verschwinden und wiederzukommen. Auf dem Papier wird aus der Kette die Rechnung.

3.6 Eine Notiz, die nur du siehst

`speaker-note` gehört zur Folie und erscheint nirgends auf der Leinwand.

```
== Das Ergebnis

#statement[$ h = 28 "m" $]

#speaker-note[
  Erst rechnen lassen, dann zeigen. Wer 28 sagt, hat gerundet -- 28,0 ist
  genauer, als die Messung hergibt.
]
```

Zu sehen ist sie in der Sprecheransicht und im Handout neben ihrer Folie.

3.7 Das Handout

Ein Argument macht aus dem Foliensatz ein Blatt zum Mitschreiben: drei Folien je Seite, die Notiz daneben, und wo keine steht, Linien.

```
#import "@preview/typstage:0.1.0": *
#show: presentation.with(
  title: [Wie hoch ist der Turm?],
```

```
  handout: 3,
)
```

Die Folien werden dabei nicht neu gesetzt, sondern verkleinert; das Handout kann also nicht von der Leinwand abweichen.

3.8 Der ganze Quelltext

Fünfundvierzig Zeilen, und nichts darin, das nicht oben erklärt wurde.

```
#import "@preview/typstage:0.1.0": *

#show: presentation.with(
  title: [Wie hoch ist der Turm?],
  author: [Klasse 9b],
  theme: themes.lesson,
)

= Die Frage

== Ein Stab und ein Turm

Ein Stab von 1,20 m wirft einen Schatten von 0,90 m.
Der Turm wirft 21 m. Wie hoch ist er?

== Was wir sehen

#stagger[
  - Die Sonne steht für beide gleich hoch.
  - Also ist der Winkel derselbe.
  - Also sind die Dreiecke ähnlich.
]

#callout[
  In ähnlichen Dreiecken stehen entsprechende Seiten im selben Verhältnis.
]

= Die Rechnung

== Das Verhältnis

#align(center, morph(<turm>, $ h / 21 = 1.2 / 0.9 $))

== Nach h aufgelöst

#align(center, morph(<turm>, $ h = 21 dot 1.2 / 0.9 $))

== Das Ergebnis

#statement[$ h = 28 "m" $]

#speaker-note[
  Erst rechnen lassen, dann zeigen. Wer 28 sagt, hat gerundet -- 28,0 ist
  genauer, als die Messung hergibt.
]
```

> **TIPP**
> Als Nächstes lohnen sich `side-by-side` – Zeichnung links, Text rechts, der häufigste Folienbau überhaupt – und `theme:`, das das Aussehen ändert, ohne dass eine Zeile des Inhalts sich rührt.

4 Eine Folie Schritt für Schritt aufdecken

Eine Folie, die sich vor der Klasse entfaltet, statt fertig dazustehen: Sie ist dann kein Bild, sondern ein Ablauf aus **Schritten**. Ein Tastendruck zeigt den nächsten Stichpunkt, der übernächste die Formel darunter; erst wenn nichts mehr aussteht, blättert der Druck weiter.

4.1 Welches Mittel wofür

Sechs Bausteine decken fast alles ab; sie lassen sich auf einer Folie mischen.

Mittel	Wofür
`#pause`	Die Folie entfaltet sich von oben nach unten. Der häufigste Fall.
`stagger[…]`	Eine Liste Punkt für Punkt – Aufzählungszeichen und Text gemeinsam. Auch für mehrere Blöcke nacheinander.
`anim(…)`	Ein bestimmtes Stück auf einem bestimmten Schritt, mit eigener Bewegung – überall dort, wo `#pause` nicht hinreicht: in Rasterzellen, Tabellen, Kästen.
`alternatives(…)`	Mehrere Fassungen derselben Sache an derselben Stelle, jede die vorige ersetzend.
`build(…)`	Eine Zeichnung oder ein Diagramm, das in Stufen entsteht – eine CeTZ-Linie, eine lilaq-Datenreihe, eine Beschriftung nach der anderen.
`scene(…)`	Eine Zeichnung, die von einem Wert abhängt, und die Werte, an denen der Vortrag hält. Für alles, was sich **bewegt**, statt dazuzukommen.

Dazu kommt `tiles` für ein Kachelraster, das sich von selbst staffelt (Kapitel „Das eigene Aussehen“), und `morph` für Objekte, die zwischen zwei Folien fliegen (Kapitel „Eine Rechnung entwickeln“).

4.2 Der Schrittzeiger

Jede Folie führt einen Schrittzeiger mit. `at` ist vorgabemäßig `auto`: der nächste freie Schritt. Aufeinanderfolgende Einblendungen nummerieren sich damit von selbst; meist steht in einer Folie überhaupt keine Zahl.

```
#anim[Der Zeiger fängt bei eins an.]   // Schritt 1
#stagger[
  - Die Liste zählt weiter,           // Schritt 2
  - wo die Folie stand,               // Schritt 3
  - Punkt für Punkt.                  // Schritt 4
]
```

Der Zeiger fängt bei eins an.

- Die Liste zählt weiter,
- wo die Folie stand,
- Punkt für Punkt.

Der Zeiger beginnt auf jeder Folie neu; Schrittnummern sind folienlokal. Eine ausgeschriebene Zahl setzt ihn neu, und von dort zählt er weiter:

```
#anim[zuerst]          // 1
#anim(at: 4)[spät]     // 4
#anim[danach]          // 5
```

Die Schreibweisen von `at`:

Geschrieben	Bedeutung
`auto`	der nächste freie Schritt (Vorgabe)
`3`	ab Schritt drei – dasselbe wie `"3-"`
`(2, 5)`	auf Schritt zwei und auf Schritt fünf
`"2-"`	ab Schritt zwei
`"1-2"`	auf Schritt eins und zwei, danach nicht mehr
`"2,4"`	auf Schritt zwei und auf Schritt vier
`"-2"`	von Anfang an bis Schritt zwei
`"3"`	genau auf Schritt drei

Eine bloße Zahl ist ein offenes Ende: Was einmal da ist, bleibt bis zum Ende der Folie. Das ist der Regelfall. Eine geschlossene Angabe wie `"1-2"` oder `"3"` lässt das Element wieder verschwinden – dann greift `exit`.

4.3 Eine Folie ohne eine einzige Zahl

Für eine Folie, die sich nur entfaltet, muss nichts umhüllt werden. `#pause` schiebt alles Folgende einen Schritt weiter:

```
== Wie eine Ableitung entsteht

Zwei Punkte auf dem Graphen, dazwischen eine Sekante.

#pause

Rückt der zweite Punkt an den ersten heran, wird aus der Sekante
eine Tangente.

#pause

Ihre Steigung ist die Ableitung an dieser Stelle.
```

Die Läufe zwischen den Pausen werden durchnummeriert: Der erste steht von Beginn an, der zweite kommt auf Schritt zwei, der dritte auf Schritt drei. Danach zählt der Zeiger weiter – ein `stagger` unter zwei Pausen beginnt bei vier.

> **ACHTUNG**
> `#pause` wird nur auf der obersten Ebene des Folienrumpfs gelesen, `#set`- und `#show`-Regeln eingeschlossen. In einer Rasterzelle, einer Tabelle oder einer Abbildung wird sie **nicht** gesehen. Dort ist `anim` das Mittel der Wahl.

```
#set text(size: 20pt)
Erster Lauf.
#pause                        // gesehen
Zweiter Lauf.

#grid(columns: (1fr, 1fr),
  [links #pause danach],      // ungesehen
  [rechts],
)
```

4.4 Eine Liste Punkt für Punkt

`stagger` staffelt die Punkte einer Liste über die Schritte. `start` ist `auto` und schließt an den Zeiger an.

```
#stagger[
  - Aufzählungszeichen und Text gehören zusammen
  - und erscheinen deshalb gemeinsam
  - Punkt für Punkt, einer je Schritt
]
```

- Aufzählungszeichen und Text gehören zusammen
- und erscheinen deshalb gemeinsam
- Punkt für Punkt, einer je Schritt

Nummerierte Listen funktionieren genauso. Ohne Liste im Rumpf wird dieser als ein Stück eingeblendet; mehrere Argumente staffeln beliebige Blöcke:

```
#stagger(
  card(title: [Behauptung])[Die Innenwinkelsumme beträgt $180°$.],
  card(title: [Begründung])[Parallele durch einen Eckpunkt, Wechselwinkel.],
  card(title: [Beispiel])[$70° + 60° + 50° = 180°$],
)
```

BEHAUPTUNG

Die Innenwinkelsumme beträgt 180°.

BEGRÜNDUNG

Parallele durch einen Eckpunkt, Wechselwinkel.

BEISPIEL

$70° + 60° + 50° = 180°$

Argument	Wirkung
`start`	erster Schritt; `auto` schließt an den Zeiger an
`stride`	Abstand der Schritte; `2` lässt je einen Schritt aus, `0` setzt alle Punkte auf denselben
`stagger`	zusätzliche Verzögerung je Position, in Millisekunden (Vorgabe 60)
`enter`	Bewegung der Punkte, wie bei `anim`
`spacing`	Abstand zwischen den Zeilen (Vorgabe 0.65 em)
`dim`	`true` lässt jeden Punkt gedämpft stehenbleiben, sobald der nächste kommt (Vorgabe `false`)
`duration`	Dauer je Eintritt in Millisekunden
`easing`	Kurve je Eintritt
`morph`	statt einzublenden, wächst jede Zeile aus der vorherigen heraus – der Weg für eine Umformungskette. `enter`, `easing` und `dim` weist er dann zurück
`name`	Name der Gruppe, nötig nur für `stagger-layer`

`dim: true` macht aus der Staffelung einen Gang: Der Punkt, über den gerade gesprochen wird, steht da, die vorherigen bleiben lesbar, aber gedämpft.

```
#stagger(dim: true)[
  - Was der Raum schon weiß
  - Was er gleich lernt
  - Was er danach kann
]
```

Jeder Punkt hält dabei nur seinen eigenen Schritt und ruht danach gedimmt (siehe „Der gedimmte Ruhezustand“). Auch der letzte Punkt dimmt, sobald die Folie nach ihm noch einen Schritt hat.

`stride: 0` legt alle Punkte auf einen Schritt; zusammen mit `stagger` kommen sie dann kurz nacheinander – eine Welle statt einer Folge.

```
#stagger(stride: 0, stagger: 60)[
  - alle drei auf Schritt eins
  - der zweite 60 ms später
  - der dritte 120 ms später
]
```

4.4.1 Was neben einem Stück steht

`stagger-layer` hängt etwas an den Schritt eines bestimmten Stückes – etwa die Umformung am Rand einer Rechnung, die zu der Zeile gehört, aus der die nächste hervorgeht. Dafür braucht die Staffelung einen Namen: `name:` sagt ihn, ein `morph:`-Name tut es auch.

```
#stagger(morph: "umformung",
  $ x^2 + 6x + 2 = 0 $,
  $ x^2 + 6x = -2 $,
  $ (x + 3)^2 = 7 $,
)

#stagger-layer("umformung", 2)[$| -2$]
```

Die Schicht bleibt von ihrem Stück an stehen. Sie trägt keinen Morph-Namen, fliegt also nicht mit, sondern erscheint nur daneben.

Die Staffelung muss im Quelltext **vor** ihren Schichten stehen; eine Schicht liest nach, welchen Schritt ihr Stück bekommen hat.

> **ACHTUNG**
> `spacing:` gilt nur für den Listenzweig. Stehen die Stücke einzeln da, zählt der gewöhnliche Blockabstand.

4.5 Aufdecken in der Reihenfolge, in der es genannt wird

Manche Stichpunkte haben keine Ordnung: Was ein Graph zeigt, was an einem Versuch auffällt, welche Rechenwege es gibt. Die Klasse nennt das in beliebiger Folge. Bei `cue` decken deshalb die Ziffern `1` bis `9` auf, was gerade genannt wurde.

```
#cue("ablesen", start: 2)[
  - positive und negative Werte
  - tiefster und höchster Wert
  - Abnahme und Zunahme
]
```

Die Gruppe braucht einen Namen, damit `cue-layer` sie findet. Sie verbraucht so viele Schritte, wie sie Punkte hat; die Reihenfolge ändert daran nichts. Beim Setzen behält die Liste ihre Leserichtung – ein noch ungenannter Punkt hält seinen Platz frei, damit nichts springt.

4.5.1 Was mit dem Punkt zugleich erscheint

`cue-layer` hängt etwas an denselben Schritt – eine Zeichenschicht, ein Bild, einen Satz daneben:

```
#cue-layer("ablesen", 1, [dazu das Passende])
```

Punkt und Schicht teilen sich einen Schritt; wer den Schritt verschiebt, bewegt beide. An einen Punkt darf beliebig viel hängen.

Die Gruppe muss im Quelltext **vor** ihren Schichten stehen; sonst meldet das Paket einen Fehler.

> **TIPP**
> Für eine CeTZ-Zeichnung, die mit den Punkten wächst, ist jede Schicht eine eigene, vollständige Zeichnung: Alles andere bleibt über `cetz.draw.hide(rest, bounds: true)` unsichtbar, zählt aber für den Ausschnitt. So liegen alle Schichten deckungsgleich übereinander und der Graph steht still.
>
> Eine Schicht trägt dabei **nur ihren eigenen Beitrag**, kein Gitter und keine Grundkurve – sonst übermalt die zuletzt gesetzte Schicht die erste. Soll die Zeichnung in der geschriebenen Reihenfolge wachsen, nimmt man `build`.

> **HINWEIS**
> Der Pfeil nach rechts deckt den nächsten **noch ungenannten** Punkt auf, in der geschriebenen Reihenfolge; wer eine Ziffer drückt, bekommt diesen Punkt. Beides mischt sich frei. Erst wenn die Gruppe voll ist, führt der Pfeil weiter. Ein Schritt zurück gibt den zuletzt genannten Punkt wieder frei.
>
> In der Sprecheransicht steht jeder noch offene Punkt blass da, mit seiner Ziffer auf dem Aufzählungspunkt; im Saal ist er unsichtbar.

4.6 Ein einzelnes Stück auf einem eigenen Schritt

`anim` blendet beliebigen Inhalt auf bestimmten Schritten ein – dort, wo `#pause` nicht hinsieht (Rasterzellen, Tabellen, Kästen), und überall dort, wo ein Stück eine eigene Bewegung bekommen soll.

```
#grid(
  columns: (1fr, 1fr, 1fr),
  gutter: 10pt,
  anim(card[*von links* \ `enter: "fade-right"`], enter: "fade-right"),
  anim(card[*von unten* \ `enter: "rise"`], enter: "rise"),
  anim(card[*unscharf* \ `enter: "blur"`], enter: "blur"),
)
```

Im Browser kommen die drei Karten nacheinander auf den Schritten eins bis drei, jede mit ihrer eigenen Bewegung. Auf Papier stehen sie nebeneinander; der Platz ist in beiden Zielen derselbe.

4.6.1 Auftritt und Abgang

Wert	Bewegung
`"fade"`	nur Deckkraft; Vorgabe für `exit`

Wert	Bewegung
`"fade-up"`	aufblendend nach oben; Vorgabe für `enter`
`"fade-down"`	aufblendend nach unten
`"fade-left"`	aufblendend nach links, kommt also von rechts
`"fade-right"`	aufblendend nach rechts, kommt also von links
`"scale"`	wächst heran
`"scale-down"`	schrumpft zusammen
`"blur"`	schärft sich ein
`"rise"`	steigt von unten auf und wächst dabei ein wenig
`"draw"`	zeichnet sich selbst – siehe „Ein Pfad, der sich selbst zeichnet“
`"none"`	ohne Bewegung – der Inhalt ist schlicht da
`"hold"`	kein Abgang, sondern ein Warten: das Stück bleibt stehen, bis das nächste da ist. Als `enter` dasselbe wie `"none"`

Die Himmelsrichtung im Namen ist die Bewegungsrichtung, nicht die Herkunft: `"fade-right"` läuft nach rechts und kommt daher von links.

Ein Name, den es nicht gibt, ist ein Fehler beim Übersetzen.

```
#anim(enter: "fade-upp")[Vertippt.]   // Fehler beim Übersetzen
```

`enter` wirkt in beide Richtungen: Beim Zurückblättern läuft derselbe Effekt rückwärts. `exit` betrifft nur den echten Abgang – wenn ein Element beim Vorwärtsblättern aus seinem Bereich fällt.

```
#anim(at: "1-2", exit: "fade-down")[Nur für zwei Schritte da.]
#anim(duration: 900, delay: 200)[Langsam und ein wenig später.]
```

`duration` ist `auto` und übernimmt die Vorgabe der Präsentation (520). `delay` ist 0. Beide sind Millisekunden und gelten für den Auftritt; beim Zurückblättern entfällt die Verzögerung.

4.6.2 Die Kurve

Alles bewegt sich vorgabemäßig auf derselben Kurve: langsam los, zügig durch, weich aus. `easing` ändert sie für ein einzelnes Element – ein Ergebnis darf über sein Ziel hinausschießen, ein Stapel Stichpunkte gleichmäßig ankommen.

```
#anim(ergebnis, enter: "rise", easing: "out-back")
#stagger(stride: 0, stagger: 60, easing: "out-quad")[
  - erst dies
  - dann das
]
```

`easing` steht bei `anim`, `stagger`, `alternatives`, `build` und `camera`. Es gilt für Auftritt, Abgang und Dimmen des Elements – nicht für den Folienwechsel und nicht für den Flug eines Magic Move.

Name	Kurve
`"standard"`	die Kurve dieses Pakets, ausgeschrieben – dasselbe wie keine Angabe
`"linear"`	gleichmäßig, ohne Anlauf und ohne Ausklang

Name	Kurve
`"ease"`, `"ease-in"`, `"ease-out"`, `"ease-in-out"`	die vier, die die Web Animations API von sich aus kennt
`"in-quad"`, `"out-quad"`, `"in-out-quad"`	sachte
`"in-cubic"`, `"out-cubic"`, `"in-out-cubic"`	deutlicher
`"in-expo"`, `"out-expo"`, `"in-out-expo"`	scharf – fast alles geschieht an einem Ende
`"in-back"`, `"out-back"`, `"in-out-back"`	holt aus und schwingt über

`in` heißt langsam los, `out` heißt weich aus. Für einen Auftritt ist fast immer `out` das Richtige: Das Auge sieht dem Ende zu und nicht dem Anfang.

Ein Name, den es nicht gibt, ist ein Fehler beim Übersetzen. Die Meldung zählt auf, was zur Wahl steht.

```
#anim(ergebnis, easing: "out-bounce")   // Fehler beim Übersetzen
```

Die drei `back`-Kurven gehen über ihr Ziel hinaus. Auf einem Weg ist das der Rückschwung; auf der Deckkraft schneidet der Browser ab, was über 1 hinausgeht. `easing: "out-back"` auf einem schlichten `"fade"` ist deshalb nur ein schnelleres `"fade"` – es lohnt sich mit Effekten, die wandern: `"rise"`, `"scale"`, `"fade-up"`.

Federn und Sprünge – `elastic`, `bounce` – gibt es nicht.

4.6.3 Der gedimmte Ruhezustand

Ein Element, dessen Bereich endet, blendet danach mit `exit` aus und behält den Platz. `after: "dimmed"` gibt den zweiten Ruhezustand: Der Punkt bleibt stehen und wird gedämpft gezeichnet – lesbar, aber nicht mehr das, worüber gerade gesprochen wird.

```
#anim(at: "2-3", after: "dimmed")[Eine Randbemerkung.]
#anim(at: 4)[Und weiter im Text.]
```

Das Element sinkt auf 65 Prozent Deckkraft und kommt beim Zurückblättern wieder herauf; bewegt oder umgefärbt wird nichts. `after` hat zwei Werte: `"hidden"` (die Vorgabe) und `"dimmed"`.

Zwei Bedingungen prüft das Paket beim Übersetzen. Erstens braucht `after` einen Bereich, der endet: `at: auto` und `at: 3` laufen bis zum Folienende, `at: "3"` und `at: "2-3"` enden. Zweitens braucht die Folie nach dem Bereich noch einen Schritt – sonst gäbe es keinen, auf dem das Element gedämpft zu sehen wäre. Deshalb steht oben die zweite Zeile.

Auf dem Papier ändert `after` nichts, ebenso wenig wie `"hidden"`: Eine Seite zeigt alle Schritte auf einmal. Die 65 Prozent sind der kleinste Wert, bei dem gedimmter Fließtext den Kontrast 4,5 zu 1 des Kontrastvertrags noch erreicht.

> **ACHTUNG**
> Die Zusage gilt für Text in `ink` über `paper` oder `surface` – also für Stichpunkte. Was schon leise ist (`muted`, Akzentfarbe), wird durch Dimmen zu leise, und über einer eigenen `card(fill: ...)` oder über einem Bild kann der Kontrast deutlich darunter fallen.

Ein verfolgtes Element **innerhalb** eines gedimmten übernimmt das Dimmen nur, wenn es genau denselben Bereich hat; ein `anim` mit eigenem Bereich bleibt voll stehen. Früher als sein Wirt erscheint ein inneres Element aber nie.

> **ACHTUNG**

> `morph`, `video`, `embed` und `flipbook` haben `at: "1-"` als Vorgabe, einen offenen Bereich, und der passt zu keinem geschlossenen. In einem gedimmten Element bleiben sie voll stehen – eine Formel in einer gedimmten Zeile steht dann schwarz in einem grauen Satz. Wer das nicht will, gibt dem inneren Element von Hand denselben Bereich oder dimmt die Zeile nicht.

4.7 Mehrere Fassungen an derselben Stelle

`alternatives` stellt mehrere Fassungen derselben Sache an denselben Ort, jede die vorige ersetzend – für Zwischenstände, die sich austauschen statt zu bewegen.

```
#align(center, alternatives(
  $ (a + b)^2 $,
  $ (a + b)(a + b) $,
  $ a^2 + 2 a b + b^2 $,
))
```

$$a^2 + 2ab + b^2$$

Sie stehen in einem Kasten, der so groß ist wie die größte von ihnen, damit ringsherum nichts springt. Jede Fassung nimmt einen Schritt; die letzte bleibt bis zum Folienende. Auf Papier steht nur die letzte, im selben Kasten.

Argument	Wirkung
`start`	erster Schritt; `auto` schließt an den Zeiger an
`align`	Ausrichtung im gemeinsamen Kasten (Vorgabe `top + left`)
`enter`	Bewegung beim Wechsel (Vorgabe `"fade"`)
`duration`	Dauer des Wechsels in Millisekunden
`easing`	Kurve des Wechsels
`inline`	hält die Fassungen in der laufenden Zeile
`morph`	die Fassungen fliegen ineinander, statt sich abzulösen. Sie stehen an derselben Stelle, der Flug hat also keine Strecke – zu sehen ist, wie die Zeichen sich an Ort und Stelle umordnen. Das ist der Weg für eine Formel, die umgeschrieben wird.

> **TIPP**
> Unterschiedlich hohe Fassungen wirken im gemeinsamen Kasten oft unruhig, weil die kürzeren oben kleben. `align: center + horizon` setzt jede in die Mitte des Kastens, und der Wechsel wird ruhig.

4.8 Eine Zeichnung, die wächst

Eine CeTZ-Zeichnung und ein lilaq-Diagramm sind **ein** Stück, nicht viele. Was darin eine Linie und was eine Datenreihe war, ist von außen nicht zu greifen; ein `anim` um eine einzelne Linie gibt es deshalb nicht.

`build` ruft die Zeichnung einmal je Schritt und legt die Fassungen deckungsgleich übereinander: auf Stufe k steht sie so, wie sie nach k Schritten aussieht. Zu sehen ist immer genau eine.

Welches Stück wann dazukommt, sagt die Frage, die jede Stufe gereicht bekommt. Sie heißt `ab`, weil sie dasselbe sagt wie `at:` sonst:

```
#build(from => box(width: 260pt, height: 58pt, {
  place(bottom + left, line(length: 100%))
```

```
  place(bottom + left, line(angle: -90deg, length: 52pt))
  place(bottom + left, dx: 22%, rect(width: 12%, height: 22pt, fill: from(2, accent)))
  place(bottom + left, dx: 44%, rect(width: 12%, height: 36pt, fill: from(3, accent)))
  place(bottom + left, dx: 66%, rect(width: 12%, height: 50pt, fill: from(4, accent)))
}), steps: 4)
```

`from(2, accent)` gibt die Farbe zurück, sobald das zweite Stück an der Reihe ist, und sonst dieselbe Farbe mit Alpha 0. Das Achsenkreuz trägt keine Nummer und steht von Anfang an da. `steps: 4` sagt, wie viele Stufen es gibt.

4.8.1 Warum Alpha 0 und nicht weglassen

Weil ein Stück, das fehlt, den Platz mitnimmt, den es hatte: Die Zeichnung wird kleiner, ein Diagramm bekommt eine andere Achsenteilung, und beim nächsten Schritt springt alles. `stroke: none` hilft nicht – damit fällt die Geometrie ebenfalls weg. Alpha 0 lässt Maß und Pfad vollständig stehen und färbt nur.

`ab` macht daraus, was zu machen ist:

Was hineingeht	Was herauskommt
eine Farbe	dieselbe Farbe mit Alpha 0
ein Strich	derselbe Strich mit durchsichtigem Pinsel; Dicke, Strichelung und Enden bleiben, denn daran hängt das Maß
ein Wörterbuch	dasselbe Wörterbuch, in dem nur die Farben und Striche zu Luft geworden sind
Inhalt	`hide(…)` – gesetzt, aber nicht gezeichnet
ein Verlauf	nichts. Ein `gradient` hat keine Deckkraft, an der sich drehen ließe; das Paket sagt es, statt es zu versuchen

Was `ab` nicht umfärben kann, bekommt die Frage mit einem einzigen Argument: `from(3)` ist wahr, sobald das dritte Stück an der Reihe ist. In CeTZ gehört dorthin `hide(…, bounds: true)`, das ein ganzes Stück verschwinden lässt und sein Maß behält.

4.8.2 Eine CeTZ-Zeichnung

Mit `#import "@preview/cetz:0.5.2"` daneben:

```
#build(from => cetz.canvas({
  import cetz.draw: *
  line((0,0), (4,0))                          // steht von Anfang an
  line((4,0), (4,3), stroke: from(2, black))    // ab Schritt 2
  line((4,3), (0,0), stroke: from(3, 1.4pt + red))
  content((2.2, 1.8), from(4, [$c$]))
  if from(4) { circle((4,0), radius: 0.18) }
  else { hide(circle((4,0), radius: 0.18), bounds: true) }
}), steps: 4)
```

4.8.3 Ein lilaq-Diagramm

Mit `#import "@preview/lilaq:0.6.0" as lq` daneben. Eine Datenreihe wird an zwei Stellen zu Luft: an ihrer Farbe und an ihrer Beschriftung in der Legende. Die zweite ist leicht zu vergessen – der Eintrag steht sonst schon in der Legende, während seine Kurve noch fehlt:

```
#build(from => lq.diagram(
  width: 7cm, height: 4.5cm,
  legend: (position: top + left),
  lq.plot(x, messung, color: from(1, red), label: from(1, [gemessen])),
  lq.plot(x, modell, color: from(2, blue), label: from(2, [Modell])),
), steps: 2)
```

Weil die Reihe als Luft in den Daten stehen bleibt, rechnet lilaq seine Achsen über beide: Die Skala steht fest, und die erste Kurve springt nicht, wenn die zweite dazukommt.

4.8.4 Die Argumente

Argument	Wirkung
`steps`	Zahl der Stufen und damit der Schritte (Vorgabe 2)
`start`	erster Schritt; `auto` schließt an den Zeiger an
`enter`	Bewegung beim Erscheinen einer Stufe (Vorgabe `"fade"`); `"draw"` ist hier ein Fehler, siehe den nächsten Abschnitt
`duration`	Dauer in Millisekunden
`easing`	Kurve der Bewegung, siehe „Die Kurve“

Auf Papier steht nur die letzte Stufe, im Block derselben Größe. Unter „Bewegung reduzieren“ ändert sich nichts: Die Stufen blenden, sie wandern nicht.

> **ACHTUNG**
> Jede Stufe wird wirklich gesetzt. Vier Stufen heißen vier Layouts und vier SVG-Bäume in der Datei – bei einer aufwendigen Zeichnung wächst beides ebenso schnell wie beim Daumenkino. Eine Zeichnung in zwanzig Stufen ist keine gute Idee.

4.9 Ein Pfad, der sich selbst zeichnet

`enter: "draw"` lässt einen Strich **entstehen**, statt ihn einzublenden: die Feder setzt an und fährt den Pfad ab, von seinem Anfang bis zu seinem Ende.

```
#anim(schaltbild, enter: "draw", duration: 900)
#stagger(enter: "draw", stride: 1, achse, kurve, tangente)
```

`duration` gilt wie überall, aber eine Zeichnung will mehr Zeit als ein Stichpunkt. 900 ist ein brauchbarer Anfang; die Vorgabe der Präsentation (520) ist für drei lange Linien knapp.

4.9.1 Was sich abfahren lässt und was nicht

Text nicht. Typst setzt Glyphen als gefüllte Umrisse ohne Kontur, und eine Fläche hat keine Länge, an der entlang etwas zu fahren wäre. Dasselbe gilt für alles Gefüllte: eine Pfeilspitze, ein ausgefüllter Punkt, die Fläche einer Karte.

Deshalb ist `draw` zweierlei zugleich: **Die Striche zeichnen sich, alles übrige blendet auf.** Die Beschriftung einer Zeichnung kommt also, während die Linien entstehen, und steht mit ihnen zusammen fertig da.

Ein Element, an dem sich **gar nichts** abfahren lässt, blendet vollständig und meldet das in der Konsole des Browsers.

4.9.2 Alle zugleich, und wie man sie nacheinander bekommt

Alle gestrichenen Pfade eines Elements fahren **zugleich** los, in der Malreihenfolge von Typst. Wer eine Reihenfolge will, gibt jedem Stück seinen eigenen Schritt:

```
#stagger(enter: "draw", stride: 1, achse, kurve, tangente)
```

4.9.3 Wo eine Zeichnung stehen muss

Nicht auf dem ersten Schritt ihrer Folie. Wer eine Folie betritt, sieht keine Auftritte; die Laufzeit stellt beim Folienwechsel nur den Zustand her. Eine Zeichnung auf Schritt eins stünde also einfach da. Sie braucht einen Schritt vor sich:

```
#anim[Erst der Satz, der die Zeichnung ankündigt.]
#anim(schaltbild, enter: "draw", duration: 900)
```

Das gilt für jeden Effekt; bei `draw` fällt es nur besonders auf.

4.9.4 Wer Konturen liefert

Die Regel ist einfach: **Was in Typst einen `stroke` bekommt, wird zu einem Pfad mit Kontur und lässt sich abfahren; was eine `fill` bekommt, nicht.** Ein Zeichenpaket liefert also genau so viel, wie es strichelt.

Eine CeTZ-Zeichnung aus wenigen langen Linien ist der Fall, für den `draw` gemacht ist: Ein Auge kann ihnen folgen. Ein lilaq-Diagramm bringt dagegen Gitter, Teilstriche, Rahmen und Marken mit – dutzende Pfade, die alle zugleich loslaufen und eher hereinwischen als entstehen. Für ein Diagramm ist `build` das bessere Mittel.

Gestrichelte Linien bleiben bei der Blende. Strichelung und Feder brauchen dasselbe SVG-Attribut. Eine gestrichelte Hilfslinie blendet deshalb auf, während ihre durchgezogenen Nachbarn sich zeichnen.

4.9.5 In beide Richtungen, und was an den Rändern gilt

Beim **Zurückblättern** fährt die Feder heraus: Was sich gezeichnet hat, zeichnet sich zurück. Ein **Sprung** – über die Adresse, über die Übersicht, beim Neuladen – stellt dagegen den Endzustand her, die fertige Zeichnung; die **Sprecheransicht** zeigt in ihrer Vorschau ebenfalls den Ruhezustand. Auf **Papier** steht die Zeichnung fertig da, `enter` erreicht die PDF nie.

`exit: "draw"` ist erlaubt und symmetrisch: Ein Element, das seinen Bereich verlässt, nimmt seine Striche zurück, statt zu verblassen.

4.9.6 Unter „Bewegung reduzieren“

Die Regel des Pakets lautet: **Deckkraft bleibt, Ortsveränderung fällt weg.** Bei `draw` ist das Zeichnen der Weg; ohne ihn bleibt die Blende stehen, und die Zeichnung erscheint wie jedes andere Element.

Wo die Reihenfolge der Striche selbst etwas erklärt, gehört sie zusätzlich in Worte – die liest auch, wer sie nicht laufen sieht.

4.9.7 Zusammen mit einer Zeichnung in Stufen

Beides zugleich geht nicht, und das Paket sagt es beim Übersetzen statt es zu versuchen:

```
#build(zeichner, enter: "draw")   // Fehler beim Übersetzen
```

Jede Stufe eines `build` ist die **ganze** Zeichnung. Eine Stufe, die sich selbst zeichnete, zöge bei jedem Schritt sämtliche Striche noch einmal nach – über der abtretenden Stufe, die längst dieselbe Tinte trägt. Zu sehen wäre nichts.

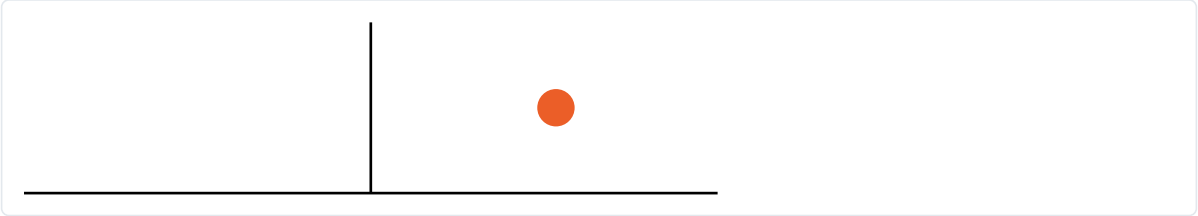
Wer eine Zeichnung Strich für Strich entstehen lassen will, gibt die Striche als eigene Stücke hin und lässt jedes sich selbst zeichnen.

4.10 Eine Zeichnung, die sich bewegt

Bei `build` kommt Stück für Stück etwas hinzu. Bei `scene` ändert sich eine **Größe**, und das Bild hängt daran.

Das Deck schreibt eine Funktion von einem Wert auf ein Bild und sagt, an welchen Werten der Vortrag hält. Typst rendert jeden Halt und die Bilder dazwischen. Ein Schritt zieht das Bild von Halt zu Halt.

```
#scene(
  x => zeichnung-bei(x),
  stops: (-3, 0, 1.5, 3),   // vier Halte, drei Schritte
  tween: 8,                 // Bilder zwischen zwei Halten
)
```



`stops` sind die Werte selbst, nicht `0.0` bis `1.0`: Wer die Tangente an der Stelle −3, im Scheitel und bei 1.5 zeigen will, schreibt diese drei Zahlen hin.

Die Szene verbraucht `stops.len() - 1` Schritte. Der erste Halt steht da, sobald die Szene erscheint; jeder weitere kostet einen Tastendruck.

4.10.1 Was zu einem Halt gehört

`scene-layer` legt einen Satz, eine Formel oder eine zweite Zeichnung auf den Schritt eines bestimmten Halts. Dafür bekommt die Szene einen Namen.

```
#scene("ableitung", x => tangente-an(f, x), stops: (-3, 0, 1.5, 3))

#scene-layer("ableitung", 2)[Im Scheitel ist die Steigung null.]
#scene-layer("ableitung", 4, enter: "scale")[$f'(x) = 1/2 x$]
```

Wer einen Halt verschiebt, verschiebt alles mit, was daran hängt. Die Szene muss im Quelltext **vor** ihren Schichten stehen; sonst meldet das Paket einen Fehler.

4.10.2 Mehrere Größen zugleich

Ein Halt darf ein Tupel sein. Der Zeichner bekommt dann ebenso viele Argumente:

```
#scene(
  (a, b) => rechteck-mit(breite: a, hoehe: b),
  stops: ((1, 1), (1, 3), (2, 3)),
  tween: 6,
)
```

Erst wächst die Höhe, dann die Breite. Was nicht geht: zwei Größen, die sich **unabhängig** voneinander bewegen. Alles reist gemeinsam von Halt zu Halt.

4.10.3 Die Argumente

Argument	Wirkung
`stops`	Die Werte, an denen der Vortrag hält. Mindestens zwei. Eine Zahl, eine Länge, ein Winkel, ein Anteil – oder ein Tupel davon.

Argument	Wirkung
`tween`	Bilder **zwischen** zwei Halten (Vorgabe 8). Mit `0` springt die Szene.
`start`	Erster Schritt; `auto` nimmt den laufenden.
`width`, `height`	Der Kasten, in dem die Szene steht (Vorgabe `100%` und `190pt`).
`duration`	Wie lange ein Zug von Halt zu Halt dauert, in Millisekunden.
`enter`	Bewegung, mit der die Szene selbst auftritt (Vorgabe `"fade"`).
`still`	Was auf Papier steht, wenn nicht der letzte Halt.
`steady`	Nachmessung der Bilder: `auto` meldet, `false` nimmt die Szene aus der Prüfung, `true` besteht darauf.

`duration` ist die Dauer des **Wegs**, nicht die der Blende, mit der die Szene auftritt.

Die Bilder einer Szene sind Zeichnungen zu verschiedenen Werten und dürfen verschieden groß ausfallen. Deshalb steht eine Szene in einem Kasten fester Größe, und jedes Bild wird darauf beschnitten. Wer `width` und `height` zu klein wählt, sieht es sofort.

> ACHTUNG
> **Der Kasten steht still, die Tinte darin nicht von selbst.** Eine CeTZ-Leinwand wächst mit ihrem Inhalt. Reicht die Tangente bei $x = -3$ weiter nach links als bei $x = 3$, sitzt das Achsenkreuz an einer anderen Stelle im Kasten – beim Blättern wandert dann das ganze Bild, obwohl sich nur ein Punkt bewegen sollte.
>
> Geradebiegen kann das Paket das nicht, bemerken schon: Jede Szene misst ihre Bilder nach und meldet abweichende Maße als Fehler beim Übersetzen.
>
> Der Ausweg liegt in der Zeichnung: ihr eine feste Ausdehnung geben und das, was sich bewegt, darin halten. In CeTZ ist das ein `rect` mit durchsichtigem Strich, dieselbe Luft, mit der `ab` arbeitet:
>
> ```
> #scene(x => cetz.canvas({
>   import cetz.draw: *
>   // Hält die Leinwand auf, egal wo der Punkt steht.
>   rect((-4.4, -0.8), (4.4, 4.6), stroke: rgb(0, 0, 0, 0))
>   line((-4, 0), (4, 0))
>   circle((x, 0.25 * x * x), radius: 0.1)
> }), stops: (-3, 0, 3), height: 160pt)
> ```
>
> Damit steht die Breite fest. Was trotzdem hinausreicht – eine Tangente etwa –, muss gekappt werden, sonst zieht sie die Leinwand doch wieder auf.
>
> **Wenn die Bilder verschieden groß sein sollen**, sagt man das: `steady: false`. Ein Rechteck, das wächst, eine Zahl, die hochzählt – dort ist der Unterschied die Sache selbst, und die Szene wird gar nicht erst gemessen. Was mit den Befunden geschieht, entscheidet `drift` an der Präsentation.

Wer sich festlegen will, schreibt `steady: true`. Dann bricht die Szene an Ort und Stelle ab, statt am Ende des Decks in einer Liste zu stehen:

```
#scene(x => cetz.canvas({
  import cetz.draw: *
  line((0, 0), (x, 0.25 * x * x))            // zieht die Leinwand mit
}), stops: (-3, 3), steady: true)            // Fehler beim Übersetzen
```

Auf Papier steht der letzte Halt; `still` setzt etwas anderes an seine Stelle. Der Schrittzeiger läuft dort trotzdem, damit `info().step.total` in beiden Ausgaben dieselbe Zahl nennt.

Unter „Bewegung reduzieren“ fallen die Zwischenbilder weg: Die Szene springt von Halt zu Halt. Siehe „Weniger Bewegung“.

4.10.4 Was eine Szene kostet

Jedes Bild ist ein echtes Typst-Layout und liegt als eigener SVG-Baum in der Datei. Übersetzungszeit und Dateigröße wachsen also mit `tween`. Gepackt fällt die Größe kaum ins Gewicht, weil die Bäume einander sehr ähnlich sind; auf Papier kostet eine Szene gar nichts, dort steht ein einziges Standbild.

> **ACHTUNG**
> Die gepackte Zahl gilt nur, solange der Webserver auch packt. Wer die Datei per USB-Stick oder als Anhang weitergibt, trägt die rohe – und die wird bei vielen Zwischenbildern megabytegroß. Die Übersetzungszeit ist immer die volle: acht Zwischenbilder je Strecke sind acht Layouts.

Das Nachmessen kostet ein weiteres Layout je Bild, nur im Browserzweig. `steady: false` gibt es einer einzelnen Szene zurück, `drift: "none"` allen. Es ist vorgabemäßig an, weil sein Befund beim Schreiben unsichtbar ist: Jedes Bild für sich sieht richtig aus, erst das Blättern zeigt die wandernde Zeichnung.

4.11 In ein Detail hineinfahren

Manchmal ist der nächste Schritt derselbe Satz aus der Nähe: das eine Feld der Tabelle, der eine Term der Gleichung, das eine Bauteil im Schaltbild. `camera` fährt darauf zu und wieder weg. Sie zielt auf ein `pin`, und auf sonst nichts.

```
#pin(<messwerk>, card(title: [Messwerk])[Thermoelement, Brücke, Verstärker.])

#camera(<messwerk>)
#anim[Und wieder heraus, im Schritt danach.]
```

Das Deck nennt damit einen Namen, keine Koordinate – wer die Folie umräumt, muss nichts nachrechnen.

4.11.1 Wie man wieder herauskommt

`at` ist ein Schrittbereich wie überall sonst: Die Folie wird genau so lange durch die Kamera gesehen, wie er gilt.

Geschrieben	Was passiert
`at: auto`	Der nächste freie Schritt, und der danach führt wieder heraus. Die Vorgabe.
`at: "3"`	Hinein auf Schritt drei, heraus auf vier.
`at: "3-5"`	Der Ausschnitt hält über drei Schritte.
`at: 3`	Hinein auf Schritt drei und bleiben; der Folienwechsel führt heraus.

Der Rückweg ist ein Schritt und wird als einer gezählt: Eine Folie mit nichts als einem Pin und einer Fahrt darauf hat drei Schritte – ganze Folie, Ausschnitt, ganze Folie.

> **HINWEIS**
> `at: auto` ist hier ein **geschlossener** Bereich, bei `anim` dagegen ein offener: Ein Auftritt hat kein natürliches Ende, eine Kamerafahrt schon. Und nie auf Schritt eins – das ist die Folie, wie man sie betritt.

4.11.2 Was mitfährt und was stehenbleibt

Gefahren wird die Folie: ihr Hintergrund und die Ebene der eingeblendeten Teile darüber. Die Folienzier fährt **nicht** mit – Fußzeile, Seitenzahl, Fortschritt und laufender Kopf stehen still und bleiben lesbar. Der Titel der Folie fährt dagegen mit; er steht im Rumpf.

Was aus dem Bild fährt, wird an der Kante der Bühne abgeschnitten. Auch die Tinte bleibt stehen, wo sie gezogen wurde: Was jemand auf die Folie malt, gehört nicht zur Folie.

4.11.3 Wie weit sie fährt

`margin` sagt, wie viel von der Folie um das Detail herum stehenbleibt, gemessen an der **unverfahrenen** Folie (Vorgabe 16 pt). Die Kamera passt Detail plus Rand ins Bild; die engere der beiden Richtungen entscheidet, damit das Ganze zu sehen ist und nicht seine Mitte. Die Fahrt dauert `duration` Millisekunden, vorgegeben 700.

```
#pin(<term>, $b^2$)
#camera(<term>, margin: 4pt, duration: 900, easing: "out-quad")
#anim[Danach.]
```

Eine Grenze nach oben gibt es nicht. Ein Pin von der Größe eines Kommas wird wandgroß gezeigt und bleibt scharf, weil Typst ihn als Vektor setzt. Ein Video, ein Bild oder eine Einbettung wird es nicht.

Ein Detail, das schon so groß ist wie die Folie, gibt nichts zu fahren.

4.11.4 Zwei Sonderfälle

Zwei Pins desselben Namens auf einer Folie. Die Kamera rahmt beide, also den Kasten um sie herum.

Zwei Fahrten, die sich auf einem Schritt überlappen. Die spätere im Quelltext gewinnt.

4.11.5 Beim Springen, beim Zurückblättern, auf Papier

Der Ausschnitt ist eine Funktion des Schritts und nichts sonst:

- **Zurückblättern** fährt den Weg rückwärts und landet sauber wieder auf der ganzen Folie.
- **Ein Sprung** – über die Übersicht, über `#3` in der Adresse, über einen Klick in der Sprecheransicht – stellt den Ausschnitt, statt ihn zu fahren.
- **Die Sprecheransicht** zeigt die laufende Folie samt Fahrt, und die Vorschau daneben trägt den Ausschnitt mit.
- Unter **Bewegung reduzieren** springt die Kamera auf den Ausschnitt, statt dorthin zu fahren.

> ACHTUNG
> **Auf Papier gibt es keine Kamera.** Das Handout setzt jede Folie ganz, und die Druckansicht des Browsers ebenso.
>
> Daraus folgt eine Pflicht für das Deck: **Die Folie muss ohne die Fahrt vollständig und lesbar sein.** Wer das Detail nur im Ausschnitt beschriftet – eine 6-Punkt-Zeile, die man ja gleich heranholt –, hat auf Papier eine Zeile, die niemand liest. Die Kamera ist eine Betonung, kein Layout.

4.11.6 Wenn der Name nicht steht

Eine Kamera, die auf ein `pin` zielt, das es auf ihrer Folie nicht gibt, ist ein Fehler beim Übersetzen und kein stummes Stehenbleiben:

```
#pin(<messwerk>, card[…])
#camera(<messerk>)          // ein Buchstabe zu wenig
```

Eine Fahrt darf vor ihrem Ziel stehen – oft gehört sie an den Kopf der Folie –, deshalb wird erst am Ende des Dokuments geprüft. Ein Pin auf der Folie **davor** zählt nicht.

Eines bleibt ungeprüft: Ein Pin, der in einem noch nicht aufgedeckten `anim` steckt, hat ein Rechteck, aber nichts Sichtbares darin. Die Kamera fährt dann auf eine leere Stelle. Welcher Schritt was zeigt, entscheidet sich erst im Browser.

4.12 Drei Stolpersteine

Nur Einblendungen zählen. Der Zeiger zählt `anim`, `stagger`, `alternatives` und `#pause`. Ein Applet, ein Video oder ein `morph` verbraucht **keinen** Schritt; solche Elemente sind von Anfang an da. In einer zweispaltigen Folie beginnen die Stichpunkte neben einem Applet deshalb bei eins:

```
#side-by-side(
  embed(url: "…", width: 100%, height: 220pt),   // kein Schritt
  stagger[
    - erster Stichpunkt                          // Schritt 1
    - zweiter Stichpunkt                         // Schritt 2
  ],
)
```

Ein Schritt vererbt sich nicht nach innen. Jedes verfolgte Element trägt seinen eigenen Schritt. Steckt eines im anderen, gilt für das innere trotzdem dessen eigene Angabe:

```
#anim(at: 3)[Ab Schritt drei -- #morph(<m>, $x^2$) aber ab Schritt eins.]
```

Bei `morph` ist das richtig so: Das **Ziel** eines Fluges muss beim Betreten der Folie schon stehen. Bei einem `anim` in einem `anim` ist es dagegen meist ein Versehen – und es fällt erst beim Blättern auf.

Ein Morph steht ab dem ersten Schritt. Weil beim Zurückblättern die Rollen tauschen, gilt das für beide Enden einer Kette. Daraus folgt: **Ein Morph gehört nicht in etwas hinein, das erst später erscheint.** Steht er in einer Kachel, die im zweiten Schritt kommt, schwebt er schon im ersten Schritt allein an dieser Stelle:

```
== In drei Schritten
#statement[#morph(<satz>, $a^2 + b^2 = c^2$)]   // steht ab Schritt eins
#tiles(card[…], card[…], card[…])               // erscheinen nacheinander
```

Für das **erste** Glied einer Kette gibt es eine Ausnahme: Dort kommt kein Flug an, die Vorfolie trägt ja keinen Morph dieses Namens. Es nimmt deshalb ein `at:` und darf mit seiner Kachel erscheinen:

```
== In drei Schritten
#tiles(
  card[Benennen …],
  card[#morph(<satz>, $a^2 + b^2 = c^2$, at: 2)],   // mit der zweiten Kachel
  card[Wurzel ziehen …],
)
```

Trägt die Vorfolie doch einen gleichnamigen Morph, ginge der Flug verloren – die Formel erschiene einfach, statt zu fliegen. Das Paket prüft das beim Übersetzen.

Ein verschachteltes Element erbt sein Einblenden. Erscheinen ein äußeres und ein inneres verfolgtes Element im selben Schritt, übernimmt das innere `enter`, `duration` und `delay` vom äußeren – sonst liefen die beiden nicht im Gleichschritt. Wer etwas Eigenes angibt, behält es; wer einen anderen Schritt wählt, erbt nichts.

Ein `fr`-Abstand gehört nicht ins verfolgte Element. `fr` ist ein Anteil an dem, was übrig bleibt, und das verteilt der Elternteil unter den Geschwistern; ein verfolgtes Element wird aber allein gemessen. Ein `#v(1fr)` unmittelbar in einem `anim` wird deshalb durchgereicht; steht es zwischen anderem Inhalt, meldet das Paket einen Fehler:

```
#anim[Links #v(1fr) Rechts]        // Fehler -- das fr gehört nach draußen
#anim[Links] #v(1fr) #anim[Rechts] // so ist es gemeint
```

Ein `fr` **innerhalb** eines Rasters ist davon nicht betroffen: `anim(grid(rows: (1fr, 1fr), …))` verteilt das Raster unter sich selbst.

5 Etwas vorführen statt behaupten

Ziel dieses Kapitels: eine Folie, auf der etwas geschieht, das Typst selbst nicht bewegen kann – eine Konstruktion, ein Video, eine gezeichnete Bewegung.

5.1 Ein Applet neben den Stichpunkten

`geogebra()` bringt GeoGebra-Applets auf die Folie. Der übliche Aufbau: links die Konstruktion, rechts die Stichpunkte, darunter die Befehle.

```
#import "@preview/typstage:0.1.0": *

#show: presentation.with(theme: themes.lesson)

== Parametervariation

#side-by-side(
  geogebra(width: 100%, height: 330pt),
  stagger[
    - Ausgangslage: $a = 1$, die Normalparabel
    - $a = "2,5"$ -- gestreckt, schmaler und steiler
    - $a = "0,35"$ -- gestaucht, breiter und flacher
    - $a = -"1,5"$ -- gespiegelt an der $x$-Achse
  ],
)

#ggb-view(x: (-3.6, 4.6), y: (-3.4, 4.6), grid: false)
#ggb-run("a=1", "f(x)=a*x^2")
#ggb-style("f", color: accent, thickness: 6)
#ggb-tween("a", at: 2, to: 2.5, duration: 950)
#ggb-tween("a", at: 3, to: 0.35, duration: 950)
#ggb-tween("a", at: 4, to: -1.5, duration: 1300)
```

`ggb-run` baut die Konstruktion auf, `ggb-style` gibt ihr die Farben der Folie, und jedes `ggb-tween` lässt einen Wert auf seinem Schritt an den neuen Wert **laufen** statt zu springen – auf denselben Schritten wie die Stichpunkte daneben. Kein Befehl nennt dabei das Applet: steht nur eines auf der Folie, finden die Befehle es von selbst. Erst zwei brauchen Namen und ein `target:`.

> **TIPP**
> Ein Tween gehört auf Schritt 2 oder später. Beim Betreten einer Folie spielt die Laufzeitumgebung alle Aufträge bis zum aktuellen Schritt **sofort** nach – ein Tween auf Schritt 1 käme deshalb nie als Bewegung an.

> **HINWEIS**
> Das Applet lädt zur Laufzeit von `geogebra.org` nach: ohne Netz bleibt der Rahmen leer. Alles Weitere zu `geogebra` und den `ggb-` Befehlen steht im Kapitel *GeoGebra*.

5.2 Was auf dem Papier an dieser Stelle steht

In der PDF bliebe an dieser Stelle ein leerer Kasten. `geogebra` und `embed` nehmen deshalb zwei Angaben für die gedruckte Ausgabe: `fallback` tritt an die Stelle des Rahmens – eine CeTZ-Zeichnung, ein Bild, eine Tabelle –, `link` setzt darunter eine im PDF anklickbare Adresse. Ohne `fallback` bleibt im Handout ein graues Rechteck mit der Beschriftung aus `label`.

```
#embed(
  url: "https://www.geogebra.org/calculator",
  width: 100%, height: 240pt,
  link: "https://www.geogebra.org/calculator",
  label: [GeoGebra-Applet],
)
```

https://www.geogebra.org/calculator

5.3 Ein eigenes Dokument einbetten

`embed` setzt beliebige Web-Inhalte in einen abgeschotteten Rahmen: `url` lädt eine Seite, `html` bettet ein eigenes Dokument als Text ein. Der Rahmen wird in Folieneinheiten vermessen und zeigt so in jedem Fenster denselben Ausschnitt.

```
#embed(
  html: "<div style=\"height:100%;display:grid;place-items:center\">"
      + "<canvas id=\"c\" width=\"320\" height=\"200\"></canvas></div>"
      + "<script>/* zeichnet in c */</script>",
  width: 100%, height: 220pt,
  fallback: align(center + horizon, [eine laufende Zeichenfläche]),
)
```

Ein Dokument aus `html` bekommt den Grundstil des Vortrags vorangestellt: es füllt seinen Rahmen, ist durchsichtig und trägt die laufende Schrift. Der eigene Stil gewinnt darüber; `style: false` schaltet den Grundstil ab.

Im gezoomten Rahmen ist ein CSS-Pixel genau ein Punkt der Folie. Wer den Inhalt in `em` bemaßt, dessen Dokument wächst mit den Folien mit; wer `15px` schreibt, hat fest 15 Punkte neben einer 19-Punkt-Folienschrift stehen.

> **ACHTUNG**
> `height: 100%` greift nur, weil der Grundstil `html` und `body` eine Höhe gibt. Mit `style: false` ist der Rahmen nur so hoch wie sein Inhalt, und `justify-content: center` zentriert im Nichts.

Soll das Dokument den Schritten der Folie folgen, bekommt es einen Namen. `bridge-job` legt einen Auftrag an diesen Namen ab, den der Browser beim Erreichen des Schritts in den Rahmen zustellt:

```
#embed(html: "…", bridge: <applet>, width: 100%, height: 240pt)
#bridge-job(<applet>, (befehl: "setze", wert: 3), at: 2)
```

`payload` ist ein Wörterbuch und wird ungelesen durchgereicht; was darin steht, ist Sache des Dokuments auf der anderen Seite. Darauf setzen die `ggb-`Befehle auf, und jedes Begleitpaket kann es genauso tun.

> **ACHTUNG**
> **Das Dokument muss sich anmelden.** An einen Rahmen, der sich nie gemeldet hat, wird nichts zugestellt, und zwar wortlos. Beide Felder werden gebraucht: alles ohne `typstage: 1` wird verworfen, bevor `ready` angesehen wird.
>
> ```
> parent.postMessage({ typstage: 1, ready: 1 }, "*");
> ```

> **ACHTUNG**
> Beim Zurückblättern und beim Betreten einer Folie wird der ganze Lauf von vorn wiederholt. Aufträge müssen deshalb wiederholbar sein: „setze a auf 2,5“ ist gut, „erhöhe a um 1“ nicht.

5.4 Video

`video` legt ein echtes HTML5-Video über die Folie: beim Betreten der Folie läuft es an, beim Verlassen hält es an.

```
#video("wellen.mp4", width: 100%, height: 240pt, poster: image("welle.png"))
```

`autoplay` und `muted` sind an, `loop` und `controls` aus – Browser lassen ein Video von sich aus nur stumm anlaufen. `radius` rundet die Ecken, `at` und `enter` sagen wie bei jedem Element, ab welchem Schritt es da ist. Auf Papier steht das `poster`; ohne `poster` bleibt im Handout ein leeres Rechteck.

5.5 Daumenkino

Bei `flipbook` zeichnet Typst jedes Einzelbild: `render` bekommt `t` von 0.0 bis 1.0 und gibt dazu das Bild – auch aus CeTZ, Fletcher oder einer Formel. Die Bilder liegen als SVG in der Datei und bleiben in jeder Größe scharf.

```
#flipbook(
  t => box(width: 100%, height: 100%,
    place(left + horizon, dx: t * 88%, circle(radius: 9pt, fill: accent))),
  frames: 24, fps: 20, width: 100%, height: 46pt,
)
```

`frames` ist die Zahl der Einzelbilder (Vorgabe 24), `fps` das Tempo beim Abspielen (Vorgabe 30). `loop` ist an; `pingpong` läuft statt dessen vor und zurück und geht dem `loop` vor. Ist beides aus, bleibt das letzte Bild stehen.

Die Uhr beginnt, wenn das Daumenkino zu sehen ist, nicht wenn seine Folie kommt: ein `flipbook(at: "3-", loop: false)` liegt auf den ersten beiden Schritten still und fängt beim Aufdecken bei null an.

Auf Papier steht ein einziges Bild: `render(0.0)`, oder was `still` an seine Stelle setzt. Steht das System des Zuschauers auf „Bewegung reduzieren“, läuft das Daumenkino gar nicht erst los.

> **ACHTUNG**
> Jedes Einzelbild wird wirklich gesetzt: 24 Bilder heißen 24 Layouts und 24 SVG-Bäume in der Datei. Bei aufwendigen Zeichnungen wächst beides schnell.

6 GeoGebra

Die Konstruktion baut GeoGebra, die Dramaturgie kommt aus den Folien. Auf jedem Schritt können Aufträge liegen: Werte setzen, Objekte zeigen oder verbergen, Farben ändern, den Ausschnitt verschieben, eine Bewegung anstoßen.

6.1 Schnellstart

`geogebra()` setzt das Applet auf die Folie; die Befehle stehen im selben Folienrumpf und geben selbst nichts aus.

```
#import "@preview/typstage:0.1.0": *

#presentation(
  slide([Ferngesteuert], {
    geogebra(app: "classic", perspective: "G", height: 240pt,
             link: "https://www.geogebra.org/calculator")
    ggb-run("a=1", "f(x)=a*x^2")
    ggb-set((a: 3), at: 2)
  }),
)
```

Die Parabel steht von Anfang an da; auf Schritt 2 wird `a` auf 3 gesetzt und sie zieht sich zusammen.

`at` ist bei allen Befehlen ein Schrittwähler wie bei `anim`: `2`, `"1-2"`, `"2,4"`, `"-2"`. Vorgabe ist `"1-"` – die meisten Aufträge richten die Konstruktion beim Betreten der Folie ein. Der Applet-Rahmen selbst verbraucht keinen Schritt.

6.2 Welches Applet gemeint ist

Ein einzelnes Applet finden die Befehle von selbst, gleich ob sie im Quelltext darüber oder darunter stehen. Zwei Applets auf einer Folie brauchen Namen, die Befehle brauchen `target`. Der Name ist eine Zeichenkette oder eine Marke:

```
#geogebra(<links>, height: 200pt)
#geogebra(<rechts>, height: 200pt)
#ggb-run("A=(0,0)", target: <links>)
#ggb-run("B=(1,1)", target: "rechts")
```

Fehlt `target` bei mehreren Applets – oder steht gar keines auf der Folie –, bricht der Bau ab:

```
error: panicked with: typstage: 2 applets on this slide
(links, rechts) — say which one is meant, e.g. target: "links".
```

6.3 Die Konstruktion aufbauen

`ggb-run` nimmt beliebig viele GeoGebra-Befehle und gibt sie einzeln an `evalCommand` weiter. Die Reihenfolge zählt: was gebraucht wird, muss vorher entstanden sein.

```
#ggb-run(at: "1-",
         "k: x^2+y^2=4", "t=Slider(0,6.283,0.01)",
         "P=(2cos(t),2sin(t))", "s=Segment((0,0),P)")
```

> **ACHTUNG**

> GeoGebras Skriptbefehle – `SetColor`, `SetValue`, `SetVisibleInView` und Verwandte – nimmt `evalCommand` **nicht** an; in `ggb-run` blieben sie wirkungslos. Dafür gibt es `ggb-set`, `ggb-style`, `ggb-show` und `ggb-hide`: sie greifen zur JavaScript-Schnittstelle, die das kann.

Was GeoGebra ablehnt, landet in der Konsole des Browsers.

Beim Betreten der Folie und beim Zurückblättern beginnt das Applet von vorn. Befehle müssen deshalb wiederholbar sein, und die Farbe gehört gleich auf `"1-"` – sonst vergibt GeoGebra beim Neuaufbau die nächste Farbe seiner Palette.

```
#ggb-run("a=1", "f(x)=a*x^2", at: "1-")
#ggb-style("f", at: "1-", color: dark, thickness: 3)
```

> **HINWEIS**
> Eine `.ggb`-Datei lässt sich nicht einbetten. Die Konstruktion entsteht mit `ggb-run` – oder sie kommt über `material` von GeoGebra: `geogebra(material: "abc123xy")`.

6.4 Werte, Aussehen, Ausschnitt

`ggb-set` nimmt ein Wörterbuch aus Objektname und Wert, `ggb-show` und `ggb-hide` beliebig viele Objektnamen. Üblich ist: alles zu Beginn aufbauen, verbergen, und Schritt für Schritt zeigen.

```
#ggb-hide("P", "s", "t", at: "1-")
#ggb-show("P", "s", at: 2)
#ggb-set((a: 3), at: 2)
#ggb-set((a: -2, b: 0.5), at: 3)
```

6.4.1 Aussehen

`ggb-style` nimmt die Objektnamen und dazu, was sich ändern soll. Was nicht genannt wird, bleibt, wie es ist.

Angabe	Wirkung
`color`	Farbe – eine Typst-Farbe, keine GeoGebra-Farbe
`thickness`	Strichstärke
`line-style`	Strichart als Zahl (durchgezogen, gestrichelt, gepunktet …)
`filling`	Füllung, 0 bis 1
`point-size`	Punktgröße
`trace`	Spur an oder aus
`label`	Beschriftung sichtbar oder nicht
`label-mode`	Art der Beschriftung als Zahl (Name, Wert, Beschriftung …)
`fixed`	gegen Verschieben festhalten
`caption`	eigene Beschriftung
`layer`	Ebene, also was vor was liegt
`position`	Ort als `(x, y)`

`color` nimmt eine Typst-Farbe: die Konstruktion trägt damit die Farben der Folien statt GeoGebras Palette.

```
#ggb-style("P", at: 2, color: accent, point-size: 6)
#ggb-style("s", at: 2, color: dark, thickness: 3)
#ggb-style("d", at: 3, color: accent, filling: 0.18, thickness: 4)
```

6.4.2 Ausschnitt

`ggb-view` setzt den sichtbaren Bereich sowie Gitter und Achsen. `x` und `y` wirken nur zusammen – beide sind Paare aus kleinstem und größtem Wert.

```
#ggb-view(at: 2, x: (-3, 3), y: (-3, 3), grid: false)
#ggb-view(at: 3, axes: false)
```

Ohne `ggb-view` ergibt sich der Ausschnitt aus `width` und `height`: ein breiter Kasten zeigt mehr x-Bereich.

6.5 Bewegung

`ggb-animate` startet GeoGebras eigene Animation. Sie läuft ohne Ende hin und her, bis die Folie verlassen wird – richtig für einen umlaufenden Punkt oder einen Schieberegler. `trace` schaltet die Spur ein, `speed` regelt das Tempo, `playing: false` hält an.

```
#ggb-animate("t", at: 3, speed: 1.2, trace: ("P",))
```

`ggb-tween` geht einmal von A nach B und bleibt dort. Der Browser zählt den Wert Bild für Bild hoch; ein Objekt, das von ihm abhängt, wächst mit – so zeichnet sich die Konstruktion selbst. `from` gibt den Anfangswert, `duration` die Dauer in Millisekunden, `easing` den Verlauf (`"ease-in-out"` oder `"linear"`). Danach sitzt der Wert auf seinem Ziel: wer zurückblättert, sieht die fertige Zeichnung.

```
#ggb-run("t_1=0", "s=Segment(A,(4*t_1,0))", at: "1-")
#ggb-tween("t_1", at: 2, to: 1, duration: 700)
```

> **ACHTUNG**
> `ggb-tween` braucht eine Schrittnummer, keinen Bereich: `at: 2`, nicht `at: "2-"`. Sonst bricht der Bau mit „`ggb-tween() needs a step number`“ ab. Und auf Schritt 1 wird nichts gezeichnet: beim Betreten der Folie setzt die Laufzeit Tweens sofort auf ihren Zielwert. Schritt 1 ist zum Aufbauen da.

6.6 Auf Papier

Im PDF gibt es kein Applet. Ohne weitere Angabe steht dort ein beschrifteter Platzhalter in der Größe des Rahmens; `link` setzt darunter einen anklickbaren Weg zum lebenden Applet.

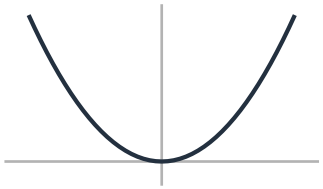
```
#geogebra(height: 90pt, link: "https://www.geogebra.org/calculator")
```

https://www.geogebra.org/calculator

Besser ist eine eigene Zeichnung. `fallback` nimmt beliebigen Inhalt – ein Bild, eine Tabelle, und vor allem eine Zeichnung mit CeTZ:

```
#geogebra(height: 120pt, link: "https://www.geogebra.org/calculator",
  fallback: cetz.canvas(length: 0.8cm, {
```

```
    import cetz.draw: *
    line((-2.6, 0), (2.6, 0), stroke: luma(70%))
    line((0, -0.4), (0, 2.6), stroke: luma(70%))
    line(..range(0, 45).map(i => (-2.2 + i * 0.1, 0.5 * calc.pow(-2.2 + i * 0.1, 2))),
         stroke: dark + 1.6pt)
  }))
```



https://www.geogebra.org/calculator

`fallback` und `link` wirken nur im PDF; im Browser steht dort das Applet.

6.7 Aussehen des Applets

Vorgabe ist `seamless: true`: das Applet trägt keinen eigenen Rahmen und seine Zeichenfläche bekommt die Farbe der Folie. `background` bestimmt diese Farbe. `auto` nimmt das Papierweiß der Präsentation – auf einer getönten Folie ist es zu ändern.

```
#geogebra(height: 240pt, background: rgb("#f4f1ea"))
#geogebra(height: 240pt, seamless: false)   // mit GeoGebras eigenem Rahmen
```

> **ACHTUNG**
> Der Ausschnitt lässt sich mit der Hand nicht verschieben. Wer im Vortrag danebengreift, schöbe sonst die ganze Ebene weg. `pan: true` gibt Verschieben und Zoomen zurück; Punkte und Schieber lassen sich in beiden Fällen ziehen.

`font-size` ist die Schrift des Applets, gezählt in Punkten der Folie; sie wächst also mit der Folie mit. Vorgabe ist 17.

> **ACHTUNG**
> GeoGebra rastet die Schriftgröße in Stufen ein: benachbarte Werte fallen oft auf dieselbe Höhe zusammen. Ein Zwischenwert gibt nicht unbedingt einen Zwischenschritt.

```
#geogebra(height: 240pt, font-size: 22)      // größere Achsenzahlen
#geogebra(height: 240pt, pan: true)          // Ausschnitt von Hand
```

`grid` und `axes` lassen GeoGebras Vorgabe stehen, solange sie `auto` sind, und erzwingen sonst das eine oder andere. `perspective: "G"` zeigt nur die Grafik-Ansicht, `app` wählt die GeoGebra-App (Vorgabe `"classic"`), `language` die Sprache der Oberfläche, `animation-button` blendet GeoGebras Abspielknopf ein.

6.7.1 Größe

`width` und `height` geben die Größe in den Maßen der Folie, nicht in Bildschirmpunkten. Welcher Ausschnitt zu sehen ist, hängt am Bereich: den setzt das Applet einmal aus den Punktmaßen des Kastens, danach gilt `ggb-view`. Eine Größenänderung lässt ihn stehen.

> **TIPP**
> Zwei Applets nebeneinander stehen am besten in einem `grid`, jedes mit `width: 100%` und eigener Höhe.

6.8 Aus der Sprecheransicht

Das Sprecherfenster führt von jedem Applet eine eigene Kopie. `m` schaltet den Zeiger vom Stift auf die Einbettung um; von da an bedient der Vortragende das lebende Applet, und die Kopie auf der Leinwand zieht nach.

Hinüber geht nur, was eine Hand bewegt: ein Punkt als seine Koordinaten, ein Schieber als sein Wert – was daraus folgt, rechnet die andere Kopie selbst aus. Wird etwas angelegt, gelöscht oder umbenannt, geht die ganze Konstruktion hinüber. Eine Animation läuft auf beiden Seiten und schickt nichts.

> **ACHTUNG**
> Ein Schrittwechsel setzt beide Kopien zurück und spielt die Aufträge der Folie erneut. Eine Änderung von Hand lebt also nur so lange wie der Schritt. Soll eine Position bleiben, gehört sie mit `ggb-set` ins Deck.

6.8.1 Die Tastatur

Ein Klick gibt dem Applet den Fokus: von da an landet jede Taste darin, bis auf die Tasten des Vortrags, die aus dem Rahmen zurückgereicht werden (siehe „Ein Rahmen, der den Fokus hat„).

> **TIPP**
> Was sich nicht bewegen soll, gehört festgehalten. `ggb-style("A", "B", fixed: true)` nagelt die Punkte fest, die eine Konstruktion nur aufspannen. Sonst greift eine Hand im Vortrag leicht den Falschen – beim Satz des Thales etwa den Durchmesser statt des Punktes auf dem Halbkreis.

`Point(k)` ist ein Punkt auf der Bahn, den eine Hand nehmen kann; `Point(k, 0.3)` liegt fest und lässt sich nicht ziehen. Wo er starten soll, sagt `position:`.

`examples/geogebra-sprecher.typ` zeigt beides: Thales mit einem Punkt auf dem Halbkreis, und eine Parabel mit zwei Schiebern.

6.9 Wessen Applet das ist

Dieses Paket schickt GeoGebra nicht mit. Es setzt einen Rahmen auf die Folie; was darin läuft, holt der Browser von `codebase`, ab Werk `https://www.geogebra.org/apps/`. Daraus folgen drei Dinge:

1. **Ohne Netz bleibt der Rahmen leer.** Wer offline vorführt, legt GeoGebras Dateien daneben und zeigt mit `codebase` darauf.
2. **Das Applet steht unter GeoGebras Bedingungen**, nicht unter der MIT-Lizenz dieses Pakets. Für eine kommerzielle Verwendung sind sie zu lesen.
3. **Der Browser des Zuschauers spricht mit `geogebra.org`.** Wo das nicht erwünscht ist – eine Klasse ohne Netz, ein Vortrag hinter einer Firewall, eine Datenschutzauflage –, lässt sich das mit `codebase` umlenken.

7 Desmos

Derselbe Weg wie bei GeoGebra, ein anderer Rechner. Alles, was das vorige Kapitel über die Brücke sagt – Schrittwähler, Zielwahl, Wiederholbarkeit – gilt hier unverändert; dieses Kapitel nennt nur, was anders ist.

Der Unterschied im Kern ist die Sprache: GeoGebra nimmt Befehle, Desmos nimmt **Ausdrücke**. Jeder trägt eine `id`, und dieselbe `id` noch einmal ersetzt den Ausdruck, statt einen zweiten anzulegen. So bewegt sich eine Kurve über die Schritte.

7.1 Der Schlüssel

`api-key` ist Pflicht. Desmos gibt sein Skript nur gegen einen Schlüssel heraus – ohne einen antwortet der Server mit 403, und der Rahmen bliebe leer.

Zum Ausprobieren gibt es `demo-key`, den Desmos in seiner eigenen Dokumentation dafür nennt. Er funktioniert, aber das geladene Skript sagt bei jedem Aufruf in der Konsole des Browsers, woran man ist:

```
This page is using the Desmos API with a trial key suitable for prototyping,
not for commercial use.
```

> **ACHTUNG**
> Wer damit vorträgt, arbeitet außerhalb dessen, wofür der Schlüssel gedacht ist. Einen eigenen gibt es über https://www.desmos.com/my-api. Und er steht im Klartext in der HTML-Datei – ein Deck, das man weitergibt oder ins Netz stellt, gibt ihn mit.

7.2 Schnellstart

```
#import "@preview/typstage:0.1.0": *

#presentation(
  slide([Eine Parabel], {
    desmos(api-key: demo-key, height: 300pt,
           expressions: (a: "a=1", kurve: "y=a x^2"),
           bounds: (-5, 5, -2, 12))
    dsm-set(("gerade": "y=2x"), at: 2)
  }),
)
```

`expressions` ist das Startbild, `bounds` der Ausschnitt als `(links, rechts, unten, oben)`. Auf Schritt 2 kommt eine Gerade dazu.

Ein Ausdruck mit `=` ist ein Regler, einer ohne eine Kurve; das entscheidet Desmos und nicht dieses Paket.

7.3 Zeigen, verbergen, entfernen

```
#dsm-hide("gerade", at: 3)
#dsm-show("gerade", at: 4)
#dsm-remove("gerade", at: 5)
```

Der Unterschied ist wichtiger, als er aussieht: Ein verborgener Ausdruck bleibt im Rechner und rechnet weiter mit – was von ihm abhängt, gilt weiterhin. Ein entfernter ist fort, und alles, was ihn nennt, geht mit ihm.

7.4 Aussehen und Ausschnitt

`dsm-style` nimmt Desmos' eigene Schlüssel. `color` nimmt eine Typst-Farbe, damit die Kurve die Palette der Folie tragen kann.

```
#dsm-style("kurve", color: rgb("#eb5e28"), line-width: 3, at: 2)
#dsm-view(bounds: (-2, 2, -1, 4), at: 3)
```

`dsm-view` verschiebt den Ausschnitt und schaltet Gitter, Achsen und Achsenzahlen. Was `desmos()` beim Aufbau bekommt, gilt für den Anfang; was `dsm-view` schickt, gilt ab seinem Schritt.

7.5 Bewegung

Zwei Wege, und sie tun Verschiedenes.

`dsm-animate` schaltet Desmos' eigene Regleranimation an. Sie läuft mit Desmos' Geschwindigkeit und ohne Ziel, hin und her, bis jemand sie anhält – richtig für ein Bild, das atmen soll.

`dsm-tween` zieht einen Regler von einer Zahl zur anderen und lässt ihn dort stehen – richtig für einen Schritt, der etwas zeigt.

```
#dsm-tween("a", to: 3.0, at: 2, duration: 900)
#dsm-animate("b", at: 4, min: -3, max: 3, step: 0.1)
```

> **ACHTUNG**
> `at` ist bei `dsm-tween` eine **Schrittnummer**, kein Wähler, und das ist Absicht. Die Bewegung liegt auf genau diesem Schritt; ab dem nächsten setzt der Befehl den Endwert einfach. Stünde sie auf „ab Schritt 2″, liefe sie bei jedem Weiterblättern der Folie neu an – gemessen sprang der Regler von 3 auf 0,75 zurück und wuchs erneut.

7.6 Auf Papier

Wie beim Applet nebenan: Der Rechner lebt nur im HTML. Im PDF steht, was `fallback` sagt, und ohne `fallback` bleibt der Platz leer. Ein Bild des fertigen Graphen ist dort meist die bessere Auskunft als ein leerer Kasten.

7.7 Wessen Rechner das ist

Der Rahmen holt Desmos von `desmos.com`, sobald die Folie gezeigt wird. Ohne Netz bleibt er leer, der Browser des Zuschauers spricht mit diesem Rechner, und was darin läuft, steht unter Desmos' Bedingungen und nicht unter der MIT-Lizenz dieses Pakets. Das PDF holt nichts.

Ein Deck, das `desmos` nie ruft, trägt von alldem nichts: Bootskript und Rahmendokument stehen hinter dem Aufruf.

8 Eine Rechnung entwickeln

Bei einer Kette von Zwischenschritten ist die entscheidende Frage, **welches Zeichen wohin gewandert ist**. Genau das leistet `morph`: dasselbe Objekt zweimal, und es fliegt von seinem alten Platz an seinen neuen.

8.1 Ein Name, zwei Folien

Mehr als ein gemeinsamer Name ist nicht nötig. Das Ding erscheint dann nicht woanders neu, sondern fliegt hinüber und nimmt die neue Größe und Gestalt an:

```
== Der Satz des Pythagoras

#align(center, morph(<pythagoras>, $a^2 + b^2$))

==

#place(center + horizon,
       morph(<pythagoras>, text(size: 2.6em)[$a^2 + b^2 = c^2$]))
```

Der Name ist eine Zeichenkette oder eine Marke: `morph("pythagoras", …)` und `morph(<pythagoras>, …)` bedeuten dasselbe. Ein Morph verbraucht keinen Schritt und steht von Anfang an auf seiner Folie. Auf Papier bleibt nur sein Inhalt.

8.2 Und auf einer Folie

Der Flug findet zwischen zwei Schritten statt – auch zwischen zwei Schritten derselben Folie. Zwei Aufrufe desselben Namens mit Bereichen, die sich nicht überschneiden:

```
== Quadratische Ergänzung

#statement[#morph(<sq>, $ x^2 + 6 x $, at: "1")]
#statement[#morph(<sq>, $ (x + 3)^2 - 9 $, at: "2-")]

#anim([Und ein Schritt, damit es einen zweiten gibt.], at: "2-")
```

Ein Morph verbraucht keinen Schritt: die Folie braucht den zweiten Schritt von woanders her, aus einem `anim` oder `stagger`. Und der Name muss auf der Folie davor frei sein, sonst ginge der Flug zwischen den Folien verloren; das Paket sagt es beim Übersetzen.

> **TIPP**
> Zwei Fassungen an derselben Stelle fliegen null Punkte weit, und man sieht nur die Zeichen sich umordnen. Wer die Bewegung sehen will, setzt die beiden Fassungen untereinander.

8.3 Zwei Abkürzungen für den häufigsten Fall

`alternatives(morph: true)` lässt seine Fassungen ineinander fliegen, statt sie zu ersetzen. Sie stehen alle an derselben Stelle: was man sieht, ist die Umformung an Ort und Stelle.

```
#alternatives(morph: true,
  $ (a + b)^2 $,
  $ (a + b)(a + b) $,
  $ a^2 + 2 a b + b^2 $,
)
```

`stagger(morph: true)` ist der nützlichere Fall: die Kette, bei der jede Zeile stehen bleibt. Die neue Zeile wächst aus der Zeile darüber, und die darüber bleibt stehen:

```
#stagger(morph: true, spacing: 14pt,
  $ x^2 + 6 x + 2 = 0 $,
  $ (x + 3)^2 - 7 = 0 $,
  $ x = -3 plus.minus sqrt(7) $,
)
```

Beide nehmen statt `true` auch einen Namen. Nötig ist er nur, wenn der Flug über den Folienrand hinaus weitergehen soll.

> **ACHTUNG**
> Ein Morph blendet nicht ein. `enter:` und `easing:` weisen beide deshalb zurück, statt sie stillschweigend fallenzulassen; `stagger` zusätzlich `dim:`, das `alternatives` gar nicht kennt. Gelesen wird `duration:`, und das ist die Dauer des Fluges.

8.4 Eine Umformungskette

Eine Rechnung über mehrere Folien folgt einem einfachen Muster: **eine Folie je Zwischenschritt**, auf jeder derselbe Morph-Name, und darunter ein Satz, der sagt, was gerade geschehen ist. Ein kleiner Helfer nimmt den immer gleichen Aufbau auf:

```
#let umform-dauer = 1500

#let umformung(gleichung, erklaerung) = {
  block(width: 100%, align(center, text(size: 1.9em, gleichung)))
  v(18pt)
  anim(block(width: 100%, align(center, text(size: 0.92em, erklaerung))),
       at: 2, enter: "fade-up")
}
```

Damit besteht jede Folie der Kette aus zwei Zeilen:

```
== Die Regel am Term -- Schritt 1 von 3
#umformung(
  morph(<term>, $f(x) = 3 x^4$, duration: umform-dauer),
  [Ein Faktor $3$ und ein Exponent $4$ -- die Potenzregel behandelt
   die beiden verschieden.])

== Die Regel am Term -- Schritt 2 von 3
#umformung(
  morph(<term>, $f'(x) = 4 dot 3 x^(4 - 1)$, duration: umform-dauer),
  [Der Exponent tritt vorn als Faktor auf -- und bleibt zugleich oben
   stehen, wo nun eins abgezogen wird.])

== Die Regel am Term -- Schritt 3 von 3
#umformung(
  morph(<term>, $f'(x) = 12 x^3$, duration: umform-dauer),
  [Erst jetzt vorn ausgerechnet: $4 dot 3 = 12$.])
```

Zweierlei ist Absicht. Die **Dauer** von 1500 ms ist länger als eine gewöhnliche Einblendung: man soll dem einzelnen Zeichen folgen können. Und die **Erklärung steht auf Schritt 2** – erst fliegt der Term, dann kommt der Satz dazu.

Auf dem Papier setzt jede Folie ihre eigene Fassung – die Kette wird zur Rechnung, Zeile für Zeile:

```
// drei aufeinanderfolgende Folien, wie sie im Foliensatz stehen
#morph(<term>, $f(x) = 3 x^4$)
#morph(<term>, $f'(x) = 4 dot 3 x^(4 - 1)$)
#morph(<term>, $f'(x) = 12 x^3$)
```

$$f(x) = 3x^4$$

Ein Faktor und ein Exponent.

$$f'(x) = 4 \cdot 3x^{4-1}$$

Der Exponent tritt vorn als Faktor auf.

$$f'(x) = 12x^3$$

Zum Schluss ausgerechnet.

> **TIPP**
> Die Folientitel der Kette durchzunummerieren („Schritt 2 von 3“) kostet nichts und hilft im Unterricht sofort: Der Fortschrittsbalken zählt Folien, nicht Zwischenschritte einer Rechnung.

Über die Höhenlage entscheidet die Quellreihenfolge, im Stillstand wie im Flug: Was nach dem `morph` notiert ist, liegt darüber, was davor steht, darunter.

8.5 Wenn die Zeichen falsch fliegen: pin

Die Paarung sucht zu jedem Zeichen der alten Folie das passende der neuen – zuerst nach der **Form**, und wo das nicht reicht, nach Nähe. Das versagt, sobald zwei gleiche Zeichen die Plätze tauschen sollen: In $f'(x) = 12x^3 - 10x + 2$ kann die Form nicht wissen, welche 2 die aus $2x$ war.

`pin` gibt einem Stück innerhalb des Morphs einen eigenen Namen. Gleiche Namen finden zueinander, bevor die Form befragt wird.

```
== Die Regel am Term -- Schritt 1 von 2
#morph(<term>, $f(x) = #pin(<faktor>)[3] x^#pin(<hoch>)[4]$)

== Die Regel am Term -- Schritt 2 von 2
#morph(<term>, $f'(x) = #pin(<hoch>)[4] dot #pin(<faktor>)[3] x^3$)
```

```
#morph(<term>, $f(x) = #pin(<faktor>)[3] x^#pin(<hoch>)[4]$)
#morph(<term>, $f'(x) = #pin(<hoch>)[4] dot #pin(<faktor>)[3] x^3$)
```

$$f(x) = 3\ x^4$$

$$f'(x) = 4 \cdot 3\ x^3$$

Ein Pin ohne Gegenstück fällt geräuschlos in den Formabgleich zurück und kostet nichts, wo er nicht gebraucht wird. Nötig ist er, wenn ein Zeichen beim Blättern stehen bleibt oder sichtbar an die falsche Stelle fliegt.

> **TIPP**
> In langen Ketten lohnt es, jedem wandernden Zeichen von vornherein einen Pin zu geben und die Namen über alle Folien der Kette durchzuhalten.

8.6 Wo der Magic Move aufhört

Drei Grenzen sind zu kennen.

Es fliegt nur von einem Schritt zum nächsten – innerhalb einer Folie oder zur unmittelbar nächsten oder vorigen. Sprünge (`o`, `Pos 1`, `Ende`, die Adresszeile) setzen das Ziel ohne Bewegung. Ein Name auf Folie 3 und derselbe auf Folie 7 tun nichts.

Zwei gleiche Namen auf der Zielfolie teilen sich dieselbe Quelle. Beide starten am selben Ort, das Zeichen spaltet sich sichtbar. Auf der Quellfolie zählt bei gleichem Namen dagegen nur der letzte.

Nicht alles ist Schrift. Wie gepaart wird, sagt `match`:

`match`	Bedeutung
`"auto"`	Zeichenweise, sofern beide Seiten Zeichen enthalten und keine von beiden mehr als 48; sonst als ein Block. Die Vorgabe.
`"glyph"`	Immer zeichenweise, auch bei vielen Zeichen.
`"block"`	Immer als ein Rechteck: Das ganze Objekt wandert und wird dabei verzerrt. Das Richtige für Bilder, Zeichnungen und alles, was keine Schrift ist.

`duration` gibt die Dauer des Fluges in Millisekunden an (Vorgabe 900). Der Wert der Zielfolie gilt, sonst der der Quelle, sonst `duration` der Präsentation.

8.7 Wie die Folie selbst wechselt

`transition` bestimmt, wie eine Folie hereinkommt. Die Präsentation setzt die Vorgabe für alle, eine einzelne Folie darf davon abweichen:

```
#show: presentation.with(transition: "slide", transition-duration: 420)

== Diese eine anders
#transition("cover", from: "bottom")

// oder, in der Argumentform:
#slide([Diese eine anders], transition: (kind: "cover", from: "bottom"))[…]
```

Art	Angaben	Was geschieht
`"none"`	–	Harter Schnitt.
`"fade"`	–	Überblendung, nichts bewegt sich.
`"slide"`	`from`	Die neue Folie rückt ein kurzes Stück heran und blendet dabei auf, die alte weicht in die Gegenrichtung.
`"push"`	`from`	Die neue schiebt die alte über den Rand hinaus.
`"cover"`	`from`	Die neue legt sich über die alte, die liegen bleibt.
`"uncover"`	`from`	Die alte zieht fort und gibt die neue frei.
`"zoom"`	`direction`	`"in"` lässt die neue nach vorn wachsen, `"out"` die alte nach hinten treten.
`"blur"`	–	Unscharf hinüber.
`"iris"`	`direction`	Eine runde Blende: `"open"` öffnet die neue Folie auf, `"close"` schließt die alte über ihr zu.
`"wipe"`	`direction`, `from`	Dasselbe als gerade Kante; `from` nennt die Kante, an der sie beginnt.

Art	Angaben	Was geschieht
`"flip"`	`axis`	Umschlagen im Raum, wie ein gewendetes Blatt.
`"cube"`	`axis`	Wie `flip`, aber als zwei Seiten eines Würfels, der sich weiterdreht.

`from` ist `"right"` (Vorgabe), `"left"`, `"top"` oder `"bottom"`. `direction` ist `"in"` / `"out"` bei `"zoom"` und `"open"` / `"close"` bei `"iris"` und `"wipe"`, jeweils der erste Wert als Vorgabe. `axis` ist `"y"` (Vorgabe, Drehung um die Senkrechte) oder `"x"`. `transition-duration` gilt für alle Übergänge (Vorgabe 420 ms).

Drei Dinge sind daran nicht selbstverständlich.

Der Übergang gehört der Grenze zwischen zwei Folien, nicht der Blätterrichtung. Maßgeblich ist die Angabe der späteren der beiden Folien.

Rückwärts läuft er als echte Umkehrung. Was hinausgeschoben wurde, kommt von derselben Seite zurück; was sich zugezogen hat, öffnet sich wieder.

Trifft ein Morph auf die Folie, überblendet sie. Fliegt zwischen zwei Folien etwas, weicht der eingestellte Übergang einer schlichten Überblendung. Für eine Umformungskette heißt das: nichts eigens abschalten.

> **ACHTUNG**
> `#transition("iirs")` und `presentation(transition: "iirs")` brechen ab und zählen die gültigen Namen auf. `slide(transition: (kind: "iirs"))` nicht – dort geht das Wörterbuch ungeprüft durch, und der Tippfehler fällt erst auf, wenn nichts geschieht.

9 Den Vortrag halten

Alles, was zwischen dem Öffnen der Datei und der letzten Folie geschieht, das zweite Fenster eingeschlossen.

9.1 Die Tasten

Die Laufzeitumgebung zählt in **Schritten**, nicht in Folien: Eine Folie mit drei Einblendungen hat drei Schritte, und `→` geht zum nächsten – gleich ob der auf derselben oder auf der nächsten Folie liegt.

Taste	Wirkung
`→`, `Bild ab`, Leertaste	einen Schritt weiter
`←`, `Bild auf`	einen Schritt zurück
`Pos 1`	zum ersten Schritt, ohne Bewegung
`Ende`	zum letzten Schritt, ohne Bewegung
`o`, `Esc`	Übersicht aller Folien ein- und ausschalten
`f`	Vollbild
`n`	die Sprecheransicht in einem zweiten Fenster öffnen
`?`	die Tastenbelegung einblenden

Ein Klick in das linke Viertel des Fensters blättert zurück, jeder andere vorwärts; in einem eingebetteten Element bleibt der Klick bei diesem. In der Übersicht führt ein Klick auf ein Vorschaubild zu dieser Folie.

Die Adresszeile trägt den laufenden Schritt mit, `#12` etwa den zwölften. Ein neu geladenes Fenster steht damit wieder an derselben Stelle, und eine von Hand geänderte Nummer springt dorthin.

9.1.1 Ein Rahmen, der den Fokus hat

Wer eine Einbettung anklickt, gibt ihr den Fokus, und von da an landet jede Taste darin. Die Tasten des Vortrags werden ihm deshalb aus dem Rahmen zurückgereicht – aber nur, wenn das Dokument sie nicht schon genommen hat, wenn der Vortrag sie überhaupt benutzt und wenn nicht in ein Textfeld getippt wird. Alles andere, `Entf` etwa, bleibt beim Rahmen.

9.2 Auf Telefon und Tablet

Ein Tippen blättert, in denselben zwei Hälften wie ein Klick. Ein Wisch blättert in der natürlichen Richtung: Der Finger schiebt die Folie nach links hinaus, also kommt die nächste. Senkrechtes Wischen und zwei Finger bleiben beim Browser – das eine ist Rollen, das andere Zoomen.

9.3 Die Sprecheransicht

`n` öffnet dieselbe Datei ein zweites Mal in einem zweiten Fenster, mit `#speaker` an der Adresse: eines für den Beamer, eines für den Vortragenden. Beide reden miteinander, auch als lokale Dateien ohne Server.

Oben stehen die beiden großen Kacheln – links die laufende Folie, rechts die Notiz –, darunter eine Zeile aus vier kleinen und einer breiten:

Kachel	was darin steht
verstrichen	die Zeit seit dem ersten Tastendruck, darunter klein die Uhrzeit an der Wand

Kachel	was darin steht
Folie	Folie / Folien, darunter Schritt / Schritte, am Fuß der Fortschrittsbalken
Ziel (min)	die geplante Dauer – `d` geht hinein –, darunter Rest und Plan, sobald eine gesetzt ist
Klassenuhr	die Uhr, die im Saal an der Wand steht; `t` startet sie
nächster Schritt	die Vorschau: was der nächste Tastendruck tut

Darunter die Werkzeugzeile: Stift oder Zeiger, die vier Farben, die Tastenbelegung. Der Zustand des Saals – `schwarz`, `eingefroren`, `kein Vortragsfenster` – steht oben rechts in der Folienkachel.

Die Tasten der Ansicht, die `?` dort auch selbst zeigt:

Taste	Wirkung
`←` `→`	ein Schritt; `Pos1` `Ende` zum ersten oder letzten
`↑` `↓`	die Notiz rollen
`o`	die Übersicht
`b`	den Saal schwarz schalten
`e`	das Bild im Saal auf diesem Schritt einfrieren
`n`	das Vortragsfenster nach vorn holen
`t`	die Klassenuhr, im Saal als Vollbild
`⇧t`	dieselbe Uhr, aber auf der Folie statt über ihr
`⇧←` `⇧→`	eine Minute weniger oder mehr, während eine Uhr läuft
`d`	die Zieldauer in Minuten
`r`	den Stundenzähler auf null zurücksetzen
`m`	zwischen Stift und Zeiger umschalten
`c`	die nächste Zeichenfarbe
`z`	den letzten Strich zurücknehmen
`x`	die Striche dieser Folie löschen
`l`	hell oder dunkel, nur für die Ansicht
`+` `-`	die Größe der Notiz
`f`	Vollbild
`?`	diese Tabelle, in der Ansicht

Auf der laufenden Folie lässt sich zeichnen; die Striche erscheinen auf der Leinwand und bleiben an ihrer Folie kleben. `x` löscht sie, `z` nimmt den letzten Strich zurück, `c` wechselt die Farbe. `b` schaltet den Saal schwarz, `e` friert das Bild auf der Leinwand ein, während man bei sich weiterblättert. `m` schaltet den Zeiger zwischen Stift und Einbettung um: im Zeigermodus landet ein Klick auf einen eingebetteten Rahmen drüben im Vortragsfenster.

> **ACHTUNG**
> Die Striche werden **nicht** mitgedruckt: die Druckansicht ist der saubere Foliensatz, nicht das Tafelbild.
>
> Schwarz und Einfrieren enden von selbst, sobald das Sprecherfenster geschlossen wird. Bleibt es offen und trägt nur kein Deck mehr, dauert das bis zu einer Minute.

9.3.1 Was die Ansicht zeigen soll

Eine Kachel, mit der niemand arbeitet, nimmt Platz, der der Notiz fehlt. `speaker-view` bestellt ab, was nicht gebraucht wird; was dort nicht steht, bleibt an:

```
#show: presentation.with(speaker-view: (
  clock: false,                          // keine Klassenuhr
  target: false,                         // keine geplante Dauer
  pen: (colors: (red, green, rgb("#FF99DD"))),   // eigene Stiftfarben
))
```

`tools: false` nimmt die Werkzeugzeile weg. Eine abbestellte Kachel nimmt ihre Tasten mit: mit `clock: false` tun `t` und `⇧t` nichts mehr. Die Stiftfarben sind Typst-Farben, keine Zeichenketten, und es dürfen mehr oder weniger als vier sein.

9.3.2 Hell oder dunkel

Die Ansicht folgt der Systemeinstellung des Rechners, an dem sie steht (`prefers-color-scheme`); `l` widerspricht ihr, und die Wahl übersteht ein Neuladen. An der Palette des Decks hängt sie **nicht**: die sagt, wie die Wand aussieht, nicht das Pult vor dem Vortragenden.

9.3.3 Eine Uhr, die die Klasse sieht

`t` fragt nach einer Zahl in Minuten, und danach steht auf der Leinwand nichts als eine Uhr: weiße `m:ss` -Ziffern auf Schwarz, aus der letzten Reihe zu lesen. Sie ersetzt die Folie und ist für die Minuten gedacht, in denen die Klasse etwas tut und nicht zuhört.

Taste	Wirkung
`t`	nach Minuten fragen; `Eingabe` startet, `Esc` lässt es
`t` (während sie läuft)	Uhr beenden, Folie wieder da
`⇧→`, `⇧←`	eine Minute mehr oder weniger, auch während sie läuft
`→` (oder jede andere Blättertaste)	beendet sie und deckt die Folie auf

Bei null hört sie nicht auf, sondern geht auf `+0:01` weiter; die Ziffern nehmen die Signalfarbe an, und über ihnen erscheint das Wort „Überzeit„.

> **ACHTUNG**
> `t`, wenn sonst nichts an der Wand steht. Keine Uhr, solange Sie reden – eine Uhr neben einem Satz zieht den Blick den ganzen Vortrag lang.

> **HINWEIS**
> Fällt das Sprecherfenster weg, hebt der Vortrag die Uhr von selbst auf. Ein Neuladen des Vortragsfensters bringt sie zurück – weiter, nicht von vorn.

9.4 Weniger Bewegung

Wer im Betriebssystem „Bewegung reduzieren“ eingeschaltet hat, bekommt ein ruhigeres Deck. Die Laufzeit fragt `prefers-reduced-motion: reduce` bei jedem Schritt neu ab; einzustellen gibt es dafür nichts. Die Regel lautet: **Deckkraft bleibt, Ortsveränderung fällt weg.**

Was	Was daraus wird
Einblendungen	`fade-up`, `fade-down`, `fade-left`, `fade-right`, `scale`, `scale-down`, `rise` und `blur` werden zur schlichten Über-

Was	Was daraus wird
	blendung. `fade` und `none` bleiben, wie sie sind. `duration` und `delay` ändern sich nicht.
`enter: "draw"`	Die Feder hält still, die Blende bleibt.
Folienübergänge	Jede Art außer `none` wird zur Überblendung, in derselben `transition-duration`. `none` bleibt der harte Schnitt.
Magic Move	Fällt aus. Es fliegt nichts, und die Folie wechselt so, wie sie es ohne Morph täte.
Daumenkino	Steht auf einem Bild still: ohne `loop` und ohne `pingpong` auf dem letzten, mit `loop` oder `pingpong` auf dem ersten. `still` gilt dabei nicht – das Bild fürs Papier steht gar nicht in der HTML.
`scene`	Springt von Halt zu Halt. Die Zwischenbilder liegen weiter in der Datei, werden aber nicht gezeigt.
`after: "dimmed"`	Bleibt. Ein Punkt, der zurücktritt, ändert seine Deckkraft und rührt sich nicht von der Stelle.
Fortschrittsbalken der Sprecheransicht	Springt auf seine neue Breite, statt hinzugleiten.

Zwei Dinge bleiben unangetastet. **Video** ist Inhalt, keine Verzierung; wer nicht will, dass es von selbst anläuft, schreibt `autoplay: false`. Und in **eingebettete Dokumente** greift die Laufzeit nicht hinein. Die Einstellung erreicht sie trotzdem: Auch dort ist `matchMedia("(prefers-reduced-motion: reduce)").matches` wahr. Wer dort etwas animiert, schreibt seine eigene `@media`-Regel.

> **HINWEIS**
> Es gibt keinen Schalter, mit dem ein Deck die Einstellung überstimmt. Wo eine Bewegung wirklich das Argument trägt, gehört sie zusätzlich in Worte – die liest auch, wer sie nicht laufen sieht.

10 Aus einer Quelle drei Ausgaben

Aus derselben Datei entstehen die Präsentation für die Leinwand, der Foliensatz zum Nachlesen und das Handout zum Mitschreiben.

10.1 Der Foliensatz

Der PDF-Lauf ergibt ohne weitere Angabe eine Seite je Folie, in der Größe der Leinwand. Jedes Element steht in seinem Endzustand: Was eingeblendet wird, ist da; von mehreren Fassungen an derselben Stelle steht die letzte. Was allein zur Bewegung gehört – Notizen, Übergänge, Aufträge an eingebettete Elemente –, fällt weg.

10.2 Das Handout

Ein einziges Argument macht aus dem Foliensatz ein Handout auf A4:

```
#show: presentation.with(handout: 3)   // drei Folien je Seite
```

`handout` nimmt `true` (zwei je Seite) oder eine Zahl von 1 bis 6 und wirkt nur auf die PDF. Die Folien werden nur verkleinert, nicht neu gesetzt: Ein Handout kann nicht von dem abweichen, was auf der Leinwand stand.

10.2.1 Alle drei Ausgaben in einem Lauf

`bundle` schreibt alle drei auf einmal:

```
#bundle(
  theme: themes.lesson,
  title: [Completing the Square],
  handout: "handout.pdf",
)[
  = Ein Abschnitt
  == Eine Folie
  Text.
]
```

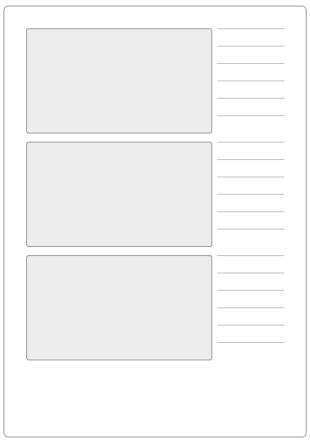
```
typst compile --features bundle,html --format bundle vortrag.typ ausgabe
```

`html`, `slides` und `handout` sind Dateinamen, `none` lässt die jeweilige Ausgabe weg, `per-sheet` sind die Folien je Handout-Blatt. Alles Übrige geht unverändert an `presentation`. Die Zähler fangen je Ausgabe neu an.

> **ACHTUNG**
> Das Bündel ist bei Typst ausdrücklich experimentell und ohne die Schalter `--features bundle,html` nicht zu haben. Und eine Datei, die `bundle` benutzt, lässt sich **nur** mit `--format bundle` übersetzen; ein gewöhnliches `typst compile vortrag.typ vortrag.pdf` bricht mit „constructing a document is only supported in the bundle target“ ab. Wer beide Wege offenhalten will, legt den Rumpf in ein `#let` und ruft `presentation` von Hand.

Neben oder unter jeder Folie steht ihre Notiz; wo eine Folie keine hat, treten Schreiblinien an ihre Stelle. Bis zwei Folien je Seite stehen sie **darunter**, ab drei **daneben**.

`handout: true`



`handout: 3`

> **TIPP**
> Titel- und Abschnittsfolien belegen auf dem Handout einen eigenen Platz. Wer viele Abschnitte führt, rechnet sie beim Blattverbrauch mit.

10.3 Notizen

`speaker-note` legt eine Notiz zur Folie ab. Sie steht im Folienrumpf oder als Angabe `note` an `slide`:

```
== Der Satz des Pythagoras
#speaker-note[
  Erst die Zerlegung zeigen, dann die Formel -- nicht umgekehrt.
]
```

Die Notiz erscheint in der Sprecheransicht – dort geht nur der reine Text ein, Auszeichnungen fallen weg – und im Handout bei ihrer Folie.

Eine Notiz muss Text tragen. Eine, die nur aus Layout besteht (ein `fit`, ein blankes `rect`, ein Bild), wird mit einer Meldung abgewiesen. Was **gesehen** werden soll, gehört auf die Folie.

10.4 Zwei Uhren für die Klasse

`t` startet die **Vollbilduhr**. Sie deckt die Folie zu, von Rand zu Rand, mit Ziffern für die letzte Reihe: Der Saal macht Pause. Blättern beendet sie.

`⇧T` startet die **angeheftete Uhr**. Sie steht *auf* der Folie und lässt die Aufgabe darunter stehen; Blättern beendet sie nicht. In der Sprecheransicht lässt sie sich mit der Maus verschieben und wandert im Vortragsfenster mit.

Beide fragen zuerst nach den Minuten und laufen erst danach. `⇧←` und `⇧→` geben eine Minute mehr oder weniger; derselbe Tastendruck noch einmal beendet die Uhr.

`class-clock` schreibt ins Deck, wie lange die Aufgabe gedacht war:

```
#slide[
  = Gruppenarbeit
  #class-clock(12)
  Sammelt zu zweit drei Beispiele.
]
```

Gestartet wird dadurch nichts: `⇧T` bietet die zwölf Minuten an, die Lehrkraft bestätigt oder ändert sie, und erst dann läuft die Uhr.

10.5 Was auf dem Papier fehlt – und was man dafür vorsieht

Auf der Folie	Auf dem Papier
`anim`, `stagger`, `#pause`	alles sichtbar, an derselben Stelle und im selben Platz
`alternatives`	nur die letzte Fassung, im gemeinsamen Kasten
`morph`	der Inhalt der jeweiligen Folie – die Kette wird zur Rechnung
`embed`, `geogebra`	`fallback`, sonst ein Platzhalter mit `label`; darunter `link`
`video`	das `poster`, sonst eine graue Fläche
`flipbook`	ein einziges Bild: `still` oder `render(0.0)`
`scene`	ein einziges Bild: `still` oder der letzte Halt
`speaker-note`	im Handout bei der Folie, im gewöhnlichen Foliensatz nichts
`transition`, `bridge-job`	nichts – sie gehören allein zur Bewegung

> **TIPP**
> Wer weiß, dass ein Handout entstehen wird, setzt `fallback` und `link` gleich beim Schreiben der Folie. Nachträglich muss jede eingebettete Stelle noch einmal aufgesucht werden.

10.6 Die HTML weitergeben

Zur HTML-Ausgabe gehören zwei Dateien: eine Stilvorlage und die Laufzeitumgebung, die die Bewegung ausführt. Wo sie herkommen, sagt `assets`:

```
#show: presentation.with(assets: "inline")                    // Vorgabe
#show: presentation.with(assets: "split")
#show: presentation.with(assets: (cdn: "https://cdn.example.org/ts/"))
```

Wert	Wirkung und Anlass
`"inline"`	Beide Dateien stehen im HTML. Eine einzige Datei, die sich verschicken, auf einen Stick legen und ohne Netz öffnen lässt. Vorgabe – und für den Unterricht meist die richtige Wahl.
`"split"`	Das HTML verweist auf `typstage-0.1.0.css` und `typstage-0.1.0.js` daneben. Angebracht, wo mehrere Vorträge in einem Ordner liegen: Der Browser lädt die Laufzeit einmal für alle.
`(cdn: …)`	Dieselben Namen unter der angegebenen Adresse. Für eine Website, die viele Vorträge trägt.

Bei kleinen Decks macht die Laufzeit den größten Teil der Datei aus. Wer viele kurze Vorträge nebeneinander veröffentlicht, spart mit `"split"` deshalb spürbar: Das erste Deck zahlt sie, jedes weitere im selben Ordner nichts mehr. Die Dateinamen führen die Version mit sich, damit kein Browser einen neuen Vortrag aus einem alten Zwischenspeicher bedient.

Typst legt keine Dateien an: Bei `"split"` und beim CDN müssen die beiden einmal geschrieben werden. Ihr Inhalt steht in `runtime-files`, und der Bündel-Export gibt sie im selben Lauf aus:

```
#import "@preview/typstage:0.1.0": *
```

```
#document("vortrag.html", title: "Der Satz des Pythagoras")[
  #show: presentation.with(title: [Der Satz des Pythagoras], assets: "split")
  == Eine Folie
  Text auf der Folie.
]

#for datei in runtime-files {
  asset(datei.name, bytes(datei.content))
}
```

```
typst compile vortrag.typ ausgabe --format bundle --features bundle,html
```

Im Ordner `ausgabe` liegen danach `vortrag.html` und die beiden Dateien, auf die es verweist.

11 Das eigene Aussehen

Ziel dieses Kapitels: eine Präsentation, die nach dem eigenen Fach und dem eigenen Geschmack aussieht – ohne dass die Bewegung darunter leidet.

11.1 Ein fertiges Theme wählen

Das Aussehen steckt in einem **Theme**: Farben, Schriften, die Form des Folientitels, der Fortschritt am Rand sowie Titel- und Abschnittsfolie. Fünf sind mitgeliefert, `theme:` wählt eines aus:

```
#show: presentation.with(theme: themes.lesson)
```

Theme	Anlass
`themes.default`	Der Vortrag im hellen Saal: dunkler Titelbalken, wachsender Fortschrittsbalken. Die Vorgabe.
`themes.lesson`	Der Unterricht: eine Farbe je Sache, kein Balken, statt einer Fußzeile ein laufender Kopf mit Nummer und Abschnitt, wie im Schulbuch.
`themes.night`	Der abgedunkelte Raum: tiefer Grund, heller Satz, kühler Akzent, Fortschritt als dünne Linie oben.
`themes.plain`	So wenig wie möglich: keine Fläche, kein Fortschritt, kleiner Titel, viel Luft.
`themes.editorial`	Werkdruckpapier, Antiqua, Haarlinien – ein Buch, keine Folie.

Die fünf sind nicht dieselbe Folie in fünf Farben: Der Titel steht mal in einem Balken, mal frei, mal unter einer Linie, und der Fortschritt wächst oder fehlt ganz.

11.2 Ein Theme abwandeln

Ein Theme ist ein Wörterbuch. `+` schreibt einzelne Einträge um – der kürzeste Weg zur eigenen Schulfarbe:

```
#show: presentation.with(theme: themes.lesson + (accent: rgb("#2f7d32")))
```

Wer alles selbst bestimmen will, baut mit `theme()` eines von Grund auf. Ohne Argument kommt die Vorgabe heraus; jedes gesetzte Argument ändert eine Sache:

```
#let schule = theme(
  paper: rgb("#fbfaf6"),
  ink: rgb("#1c2126"),
  strong: rgb("#2b4c7e"),     // Titel, Kartenkopf, Abschnittsfläche
  accent: rgb("#e0762a"),     // Striche, Fortschritt, Merkkasten
  muted: rgb("#6b7280"),      // Fußzeile, Untertitel
  font: ("Source Sans 3", "DejaVu Sans"),
  size: 20pt,                 // Fließtext auf der Folie
  title-size: 26pt,
  header: "plain",
  rule-size: 3pt,
  footer: "fraction",
  progress: "tick",
)

#show: presentation.with(theme: schule)
```

Die drei Bauformen der gewöhnlichen Folie:

Eintrag	Werte
`header`	`"band"` – farbiger Balken über die ganze Breite; `"plain"` – Titel auf dem Papier, mit `rule-size` eine Linie darunter; `"run"` – Kopfzeile wie im Schulbuch: Nummer links, Abschnitt rechts, Haarlinie darunter. Sie liegt in der Ebene der Fußzeile und wandert beim Blättern nicht mit.
`footer`	`"fraction"` (3 / 12), `"number"` (3), `"center"` (mittig) oder `"none"`; `footer-rule` legt eine Haarlinie darüber.
`progress`	`"bar"` (wachsender Balken unten), `"top"` (dasselbe oben), `"tick"` (wandernde Marke auf einer Schiene) oder `"none"`.

`box` sagt, wie eine `card` gebaut ist. `"bar"` ist die Vorgabe: weiße Fläche, dünner Rahmen, farbiger Streifen mit versalem Etikett darüber. `"label"` kommt aus dem Schulbuch: keine Kante, keine Rundung, getönte Fläche, Beschriftung in der Farbe im Kasten. Die Tönung folgt der mitgegebenen Farbe; ohne eigene Farbe gilt `surface`.

Weitere Einträge steuern die Karten (`surface`, `border`), hell auf dunkel (`inverted`), die Luft um den Rumpf (`head-gap`, `foot-gap`, `band-height`) sowie `title-slide` und `section` – zwei ganze Bilder als Funktionen `(t, s, geo) => content`. Die vollständige Liste steht in der API-Referenz.

11.3 Eine Palette wählen

Ein Theme sagt, wie eine Folie **gebaut** ist; eine **Palette** sagt, welche Farbe sie hat. `presentation` nimmt die Palette getrennt entgegen, und sie überschreibt nur die Einträge, die dastehen:

```
#show: presentation.with(theme: themes.lesson, palette: (accent: blue))
#show: presentation.with(theme: themes.lesson, palette: palettes.dark)
```

Eine Palette trägt acht Einträge, genau die Farbeinträge eines Themes: `paper` der Grund der Folie, `ink` der Fließtext, `strong` die tragende dunkle Farbe, `accent` die Signalfarbe, `muted` das Nebensächliche, `surface` der Grund einer Karte, `border` deren Kante und `inverted`, ob heller Satz auf dunklem Grund steht. Einen Eintrag, den es nicht gibt, weist das Paket ab: `palette: (acent: blue)` bricht mit einer Meldung ab.

Fünf Paletten sind mitgeliefert. Jede läuft mit jedem der fünf Themes:

Palette	Woher
`palettes.light`	Die Farben von `themes.default`. Ändert an der Vorgabe nichts.
`palettes.mono`	Das Grau von `themes.plain`, zwei Töne verschoben.
`palettes.textbook`	Die Schulbuchfarben von `themes.lesson`, ein Grau verschoben.
`palettes.parchment`	Das Werkdruckpapier von `themes.editorial`, zwei Töne verschoben.
`palettes.dark`	Der dunkle Grund von `themes.night`, mit tieferem Akzent.

Daraus folgt: **ein weiteres dunkles Theme braucht es nicht, weil Dunkelheit eine Palette ist und keine Gestaltung.** `themes.lesson` mit `palettes.dark` ist weiterhin der Unterrichtsentwurf, nur dunkel. `themes.night` bleibt trotzdem ein Theme: sein Zyan leuchtet auf dem eigenen Grund, hält aber auf einer umgedrehten Folie nicht – `palettes.dark` nimmt darum ein tieferes Blau.

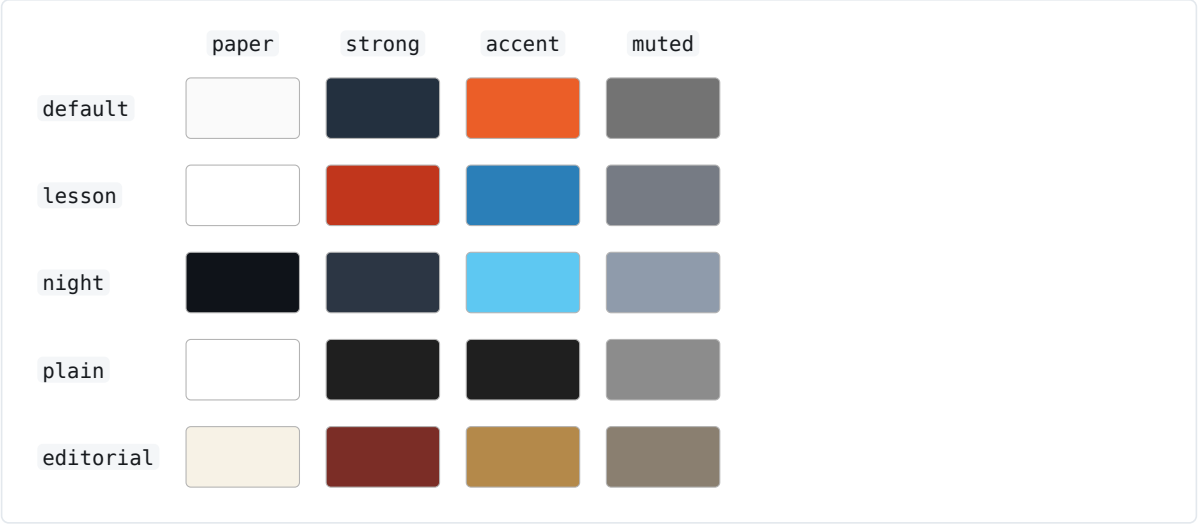
> **ACHTUNG**
> `title-fill` und `rule-fill` sind keine Paletteneinträge. Ob sie einer Palette folgen, entscheidet das Theme; alle fünf mitgelieferten lassen sie folgen, entweder als Funktion der Palette

(`title-fill: p => p.strong`) oder als `none`, was den Akzent meint. `themes.X.title-fill` **auszulesen** liefert deshalb eine Funktion, `rule-fill` liefert `none`. Wer dort eine feste Farbe **hinschreibt**, behält sie unter jeder Palette.

11.4 Die Farben eines Themes

Dieselben acht Einträge, die eine Palette trägt, belegen die fünf Themes verschieden:

```
#import "@preview/typstage:0.1.0": themes
#themes.night.accent      // die Signalfarbe des Themes, als Farbe
```



`card` und `callout` holen sich ihre Farben selbst aus dem laufenden Theme; ein Themewechsel färbt sie mit um. Eine einzelne Karte nimmt `color:` und `fill:` entgegen.

TIPP
Eine eigene Bedeutungsfarbe – blau für die Funktion, orange für ihre Steigung – legt man einmal oben in der Datei fest und gibt sie überall mit: `card(color: …)`, `callout(color: …)`, `ggb-style(color: …)`.

Unabhängig vom Theme gibt das Paket vier Farbkonstanten heraus – `dark`, `accent`, `paper` und `muted`, die Farben des Vorgabe-Aussehens. Wer das Theme wechselt, greift besser auf dessen Einträge zu.

11.5 Eine Folie umdrehen

Für die eine Folie, die nur eine große Zahl trägt, gibt es `invert`. Der Grund wird zur Schriftfarbe der Palette, der Satz zu ihrem Grund; `muted`, `border` und `surface` mischen sich aus beiden, `strong` und `accent` gehen unverändert mit. Kopf, Fuß, Foliennummer, Fortschritt, Karte und Merkkasten ziehen mit.

In der Überschriftenschreibweise steht `#invert` im Rumpf der Folie, so wie `#pause`:

```
== Erreicht bis 2026
#invert
#statement[74 %]
```

In der Argumentschreibweise ist es ein Argument von `slide`:

```
#slide([Erreicht bis 2026], invert: true)[#statement[74 %]]
```

ACHTUNG

Nur eine gewöhnliche Folie dreht sich um. Titel- und Abschnittsfolie sind ganze Bilder, die das Theme selbst malt; sie nehmen das Argument nicht an.

Die Marke `#invert` wird überall gefunden, wo der Rumpf begehbar ist – auch in Blöcken, Tabellenzellen, Rastern, in der Folienüberschrift und hinter `#set`- und `#show`-Regeln. **Nicht** gefunden wird sie, wo der Inhalt an eine Closure geht. Nachgemessen sind das neun: `context`, `fit`, `anim`, `card`, `callout`, `tiles`, `cue`, `stagger` und `alternatives`. Die Folie bleibt dann ohne Meldung stehen; wer eine davon braucht, schreibt `slide(invert: true)`.

11.6 Der Kontrastvertrag

Die mitgelieferten Paletten werden gemessen, bevor sie ausgeliefert werden. Gerechnet wird der WCAG-2-Kontrast, geprüft werden sieben Paarungen:

Paarung	Mindestens	Wofür
`ink` auf `paper`	4,5	Fließtext auf der Folie
`ink` auf `surface`	4,5	Fließtext in einer Karte
`muted` auf `paper`	4,5	Fußzeile, Untertitel, Kopfzeile
`accent` auf `paper`	3,0	Striche, Fortschritt, Marke
`accent` auf `ink`	3,0	dasselbe auf einer umgedrehten Folie
`accent` auf Schwarz	3,0	die Überzeit der Vollbilduhr
`border` auf `paper`	1,2	Haarlinien

Die vorletzte fällt aus der Reihe: ihr Grund ist keine Rolle der Palette, sondern Schwarz selbst – die Vollbilduhr ist schwarz von Rand zu Rand, was immer die Palette sagt. Geprüft wird jede der fünf Paletten **und** ihre umgedrehte Form, als `assert` beim Laden des Pakets; eine Farbe, die den Vertrag verletzt, bricht den Bau mit der Zahl, die sie verfehlt hat.

ACHTUNG
Der Vertrag gilt nur für die mitgelieferten Paletten. Eine eigene Palette wird nicht geprüft – weder gewarnt noch umgefärbt. `palette-report(…)` gibt dieselbe Messung als Liste zurück:

```
#for f in palette-report((paper: white, ink: black, surface: white,
                          muted: luma(55%), accent: blue, border: luma(86%))) [
  #f.pair: #calc.round(f.ratio, digits: 2) (will #f.min) #f.ok \
]
```

`contrast(a, b)` ist die Rechnung selbst und nimmt zwei beliebige Farben.

Und die fünf Themes bestehen ihn nicht:

Theme	Was durchfällt
`themes.default`	nichts, alle sieben Paarungen halten
`themes.lesson`	`muted` auf `paper` misst 4,25 statt 4,5
`themes.night`	`accent` auf `ink` misst 1,59 statt 3,0
`themes.plain`	`muted` auf `paper` misst 3,35 statt 4,5; `accent` auf `ink` misst 1,27 statt 3,0
`themes.editorial`	`muted` auf `paper` misst 3,51 statt 4,5; `accent` auf `paper` misst 2,84 statt 3,0

Geändert wurde keine dieser Farben: ein Wechsel hätte jedes bestehende Deck anders aussehen lassen, und was `muted` trägt, ist Nebensächliches. Wer die Zahlen einhalten will, legt die passende Palette darüber:

```
#show: presentation.with(theme: themes.editorial, palette: palettes.parchment)
```

> **ACHTUNG**
> **Aus der Füllfarbe wird nicht auf die Schriftfarbe geschlossen.** Ein mattes Salbeigrün wie `#aebdb3` sieht für eine Helligkeitsregel „hell“ aus, aber Weiß darauf misst 1,96 zu 1 – weit unter den 4,5, die Fließtext will. Deshalb färbt das Paket nirgends automatisch um.
>
> Die eine Ausnahme steht im Theme: Wo ein Theme `strong` als **Schrift** setzt – die Überschrift in `themes.lesson`, der Abschnittstitel in `themes.plain` –, wählt es zwischen `strong` und `ink` nach dem gemessenen Kontrast gegen den Grund.

11.7 Die Leinwand

`presentation` bestimmt das Format der Folie:

```
#show: presentation.with()                              // 16:9, die Vorgabe
#show: presentation.with(width: 800pt, height: 600pt)   // 4:3
#show: presentation.with(margin: 48pt)                  // mehr Luft
```

Ohne Angabe ist die Folie 16:9 auf A4-Breite. So trägt sie den Text in derselben körperlichen Größe wie eine Handout-Seite. `height` ergibt jedes andere Verhältnis, `margin` den Abstand zum Rand.

Alles, was das Theme zeichnet, skaliert mit der Breite mit: eine halb so breite Präsentation sieht gleich aus, nur kleiner. Anders wird das Layout nur durch das **Verhältnis**, und der Browser passt Bühne, Übersichtsbildchen und gedruckte Seiten darauf ein.

11.8 Typografie

Schrift und Schriftgröße kommen aus dem Theme (`font`, `size`, `title-font`, `title-size`). Für alles Weitere – Absätze, eigene Show-Regeln – gibt es den Haken `style`: eine Funktion, die um jeden Folienrumpf gelegt wird.

```
#show: presentation.with(
  style: it => {
    set par(leading: 0.68em, spacing: 0.85em)
    show math.equation: set text(size: 1.05em)
    it
  },
)
```

> **ACHTUNG**
> Alles, was über Größe, Farbe, Schrift, Schnitt, Lage und Sprache des Textes hinausgeht, gehört in `style` und nicht in eine `#set`-Regel im Dokument. Im Browser wird jedes bewegte Element ein zweites Mal gesetzt, in einem eigenen Rahmen, der die `#show`-Regeln der Folie nicht kennt. `style` liegt auf beidem.
>
> Für die Formen, die typstage selbst zeichnet, gibt es einen zweiten Weg: Label-Regeln vor `#show: presentation`. Sie erreichen auch Kopf, Fuß und Titelfolie. Siehe /Labels: jede gebaute Form ansprechen/ weiter unten.

> **TIPP**
> Ein Folienrumpf ist ein Kasten fester Höhe. Ein `style`, der ihn zwischen zwei Bruchteilsabstände setzt, rückt auch eine kurze Folie in die senkrechte Mitte:

```
style: it => { v(1fr); it; v(1fr) }
```

Zentriert wird mit Abständen, nicht mit `align`: Als Stilregel schlüge `align` bis in jede Rasterzelle durch, und das Aufzählungszeichen eines zweizeiligen Punktes rutschte neben dessen zweite Zeile.

11.9 Bausteine für den Folienrumpf

Sechs Bausteine für den Rumpf. Es sind Inhaltsfunktionen, keine eigenen Folienarten: sie lassen sich schachteln, in eine Rasterzelle setzen und mit `anim` einblenden.

11.9.1 card – der benannte Kasten

```
#card(title: [Potenzfunktion])[$f(x) = x^n$ mit $n in NN$.]
```

POTENZFUNKTION

$f(x) = x^n$ mit $n \in \mathbb{N}$.

`number:` setzt eine Ziffernscheibe davor – für Ablaufpläne, bei denen die Nummer zur Sache gehört. `color:` färbt den Streifen, `fill:` die Fläche.

```
#card(number: 2, title: [Zweiter Schritt])[Ableiten, dann einsetzen.]
```

ZWEITER SCHRITT

2 Ableiten, dann einsetzen.

11.9.2 callout – der Merksatz

```
#callout[Der Exponent entscheidet über die Symmetrie.]
```

MERKE

Der Exponent entscheidet über die Symmetrie.

`title:` ändert die Überschrift (Vorgabe „Merke“), `color:` die Farbe; `title: none` lässt sie weg.

11.9.3 side-by-side – zwei Spalten

Links die Zeichnung oder das Applet, rechts der Text:

```
#side-by-side(
  card(title: [Gerader Exponent])[Achsensymmetrisch zur $y$-Achse.],
  stagger[
    - $f(-x) = f(x)$
    - Wertemenge $W = [0; oo[$
  ],
)
```

GERADER EXPONENT

Achsensymmetrisch zur y-Achse.

- $f(-x) = f(x)$
- Wertemenge $W = [0; \infty[$

`split:` nimmt die Spaltenbreiten; die Vorgabe gibt der ersten etwas mehr. Mehr als zwei Spalten sind erlaubt – dann bekommen alle dieselbe Breite, sofern `split:` nicht ebenso viele Werte nennt.

`equal: true` macht alle Spalten gleich hoch; ohne das steht jeder Kasten so hoch wie sein eigener Text. Ein `height: 100%` im Kasten täte es nicht, weil ein Prozentmaß gegen die **Region** auflöst und nicht gegen die Rasterzeile; deshalb wirkt `equal` nur auf `card` und `callout`.

11.9.4 tiles – das Kachelraster

Jede Kachel erscheint einen Schritt nach der vorigen.

```
#tiles(
  card(title: [eins])[Beobachten],
  card(title: [zwei])[Vermuten],
  card(title: [drei])[Begründen],
)
```

`columns:` legt die Spaltenzahl fest (Vorgabe: bis zu drei). `stride: 0` lässt alle im selben Schritt erscheinen und staffelt nur über `stagger` in Millisekunden – dann läuft eine Welle durch das Raster:

```
#tiles(stride: 0, stagger: 90, [A], [B], [C], [D])
```

`duration:` und `easing:` sind die von `anim` und gelten für jede Kachel gleich. Ohne Angabe gilt die Dauer der Präsentation.

```
#tiles(duration: 500, easing: "out-back", [A], [B], [C])
```

11.9.5 statement – die große Aussage

```
#statement[$ a^2 + b^2 = c^2 $]
```

$$a^2 + b^2 = c^2$$

`statement` fordert die volle Breite an und zentriert darin – genau das, woran ein blankes `align(center, …)` in einem verfolgten Element scheitert.

11.9.6 fit – den Inhalt auf seinen Platz rechnen

Für das eine Stück, dessen Größe nicht im Deck steht: die breite Tabelle aus der Auswertung, das erzeugte Diagramm, die Liste aus einer Datendatei. Ohne `fit` läuft so ein Block über den Rand, und im Browser wird abgeschnitten, was übersteht.

```
== Ergebnisse der Regression
#fit(wrap: false, meine-tabelle)
```

`fit` misst den Block gegen den Platz, an dem er steht, und skaliert ihn geometrisch. Gerechnet wird beim Übersetzen, das Ergebnis steht in HTML und PDF gleich.

Erst die Breite anbieten, dann verkleinern. Der Block bekommt die volle Breite angeboten, bevor gemessen wird; ein Absatz bricht dann um, statt zu schrumpfen. Eine Tabelle oder Zeichnung ordnet sich dabei selbst um, und das ändert das Bild statt seiner Größe – `wrap: false` misst sie so, wie sie steht. Das ist die eine Angabe, die man vor dem ersten Gebrauch kennen sollte.

Es verkleinert nur. `grow: true` bläst auch auf, was kleiner ist als sein Platz; `shrink: false` lässt nur das Vergrößern übrig.

```
#fit(grow: true)[42%]
```

`width` und `height` nehmen `auto`, eine Länge oder einen Anteil. Bei `height: auto` nimmt sich der Block, was unter dem übrigen Inhalt der Folie übrig bleibt. In einem `card` wird der Kasten damit folienhoch, unten abgeschnitten, und **was nach dem `card` steht, fällt von der Folie** – dort gibt man `height:` deshalb ausdrücklich an.

> ACHTUNG
> **Keine Einblendung im `fit`.** Ein `pause` findet sich nur, indem der Folienrumpf abläuft, und ein gemessener Block ist eine Closure, die dieser Lauf nicht erreicht – die Schritte fielen ohne Meldung weg. Ein gemessener Block hat außerdem keine feste Höhe, an der ein verfolgtes Element seine Größe festmacht.
>
> `fit` bricht deshalb ab, mit Namen und Rat, für `pause`, `anim`, `stagger`, `alternatives`, `morph`, `tiles`, `video`, `embed`, `flipbook`, `build`, `scene`, `camera` und `cue` – auch dann, wenn das `fit` in einem anderen `fit` steckt. Der Ausweg: das `fit` **innerhalb** der Einblendung setzen, nicht darum herum:
>
> ```
> #anim(fit(wrap: false, meine-tabelle))   // so
> #fit(anim(meine-tabelle))                // nicht so
> ```

`speaker-note` und `bridge-job` dürfen im `fit` stehen. Umgekehrt nicht: eine Notiz, die nur aus einem `fit` besteht, trägt keinen Text und wird abgewiesen.

Die Rechnung dahinter ist von mosaic übernommen, das sie aus Touying 0.7.4 hat; Touying schreibt die Arbeit daran Andreas Kröpelin (Polylux PR 91) und ntjess zu.

11.9.7 overflow – der Prüflauf vor dem Vortrag

`fit` richtet den einen Block, dessen Größe man ahnt. `overflow` beantwortet die Frage, die man nicht Folie für Folie stellen kann: läuft irgendwo in diesem Deck etwas über seinen Platz?

```
#show: presentation.with(overflow: "error")
```

Standardmäßig aus, gedacht für einen Lauf vor dem Vortrag. Ein Bauskript muss dafür kein Deck anfassen, sondern hebt die Einstellung von der Kommandozeile an:

```
typst compile --features html --format html \
  --input typstage-overflow=error deck.typ deck.html
```

Die Eingabe hebt an, sie senkt nie ab: es gilt die strengere der beiden Angaben, `"none"` < `"record"` < `"error"`.

`"none"` — es wird nichts gemessen. Der Vorgabewert.
`"error"` — das ganze Deck wird gebaut, und dann bricht es mit **allen** Stellen auf einmal ab statt mit der ersten.
`"record"` — es baut durch und legt je Fund einen abfragbaren Datensatz ab. Typst gibt einem Paket keinen Warnkanal, `"record"` gibt also nichts aus.

Die Meldung nennt Folie, Schritt und das Maß (hier gekürzt):

```
error: assertion failed: typstage: 2 slides run over the room the body has. …
  slide 2, from step 1 at the earliest: 311.14pt too tall, 675.76pt of content in 364.61pt of
room
  slide 3, from step 2 at the earliest: 296.49pt too tall, 661.1pt of content in 364.61pt of
```

```
room
Shorten the slide, split it, or put the block that does not fit into fit(). …
```

„at the earliest“ steht da, weil eine Folie auf Schritt eins genauso hoch ist wie auf Schritt fünf: jedes verfolgte Element hält von Anfang an seinen vollen Platz. Der Schritt ist eine untere Schranke, die **Folie** stimmt immer. Auf Papier gibt es keinen Schritt, dort steht `step: 0`.

Die Datensätze holt man mit `typst eval`, und dafür muss das Deck auf `overflow: "record"` stehen – auf `"error"` bricht auch dieser Befehl ab:

```
typst eval --target html --features html --in deck.typ \
  'query(<typstage-overflow>).map(e => e.value)'
```

```
[{"slide":2,"step":1,"height":675.76,"room":364.61,"over":311.14},
 {"slide":3,"step":2,"height":661.1,"room":364.61,"over":296.49}]
```

> **HINWEIS**
> **Was die Prüfung nicht sieht.** Gemessen wird nur die Höhe: `measure` deckelt die Breite, die es meldet, bei der Breite, die es bekommt. Für zu breite Blöcke ist `fit` die Antwort.
>
> Übersehen werden ein `height: 100%` im Rumpf (es misst 0), ein `1fr` (es fällt zusammen) und alles, was außerhalb seines Layoutkastens zeichnet: `scale`, `move`, `place` mit Versatz. Titel- und Abschnittsfolien haben keinen Rumpfblock und werden nie gemessen. Umgekehrt wird ein `v()` am Ende eines Rumpfes gemeldet, obwohl es nichts zeichnet.

In der HTML kostet der Lauf merklich Zeit, auf Papier fast keine.

11.9.8 drift – der Melder für wandernde Szenen

`drift` fragt, ob eine Szene beim Blättern stillsteht. Eine Zeichnung ist so groß wie ihr Inhalt; ändert sich der Inhalt über die Halte einer `scene`, ist jedes Bild anders groß und sitzt in seinem Kasten woanders – beim Blättern wandert das ganze Bild, obwohl sich nur ein Punkt bewegen sollte. Jede Szene misst deshalb ihre Bilder nach, und `drift` sagt, was mit den Funden geschieht.

`"error"` — das ganze Deck wird gebaut, und dann bricht es mit **allen** Szenen auf einmal ab. Der Vorgabewert.
`"record"` — es baut durch und legt je Fund einen abfragbaren Datensatz ab.
`"none"` — es wird gar nicht erst gemessen.

```
#show: presentation.with(drift: "record")
```

Die Meldung nennt Folie, Schritt und die Zahlen (hier gekürzt):

```
error: assertion failed: typstage: 1 scene draws frames of different sizes. …
  slide 4, from step 1: 28 frames in 19 different sizes, up to 28.35pt apart across and
53.86pt down
```

Die Datensätze holt man wie beim Überlauf, mit `<typstage-drift>` statt `<typstage-overflow>`; auch dafür muss das Deck auf `"record"` stehen.

Dieser Melder ist an und `overflow` nicht: er kostet nur, wer `scene` benutzt, und was er findet, ist beim Schreiben unsichtbar. Gemessen wird nur im Browserzweig – auf Papier steht ein Standbild, und das wandert nicht.

> **HINWEIS**

> **Was er nicht kann.** Er sieht den Fall, er behebt ihn nicht: `measure` antwortet mit einer Größe und nie damit, **wo** die Tinte darin liegt.
>
> **Was er übersieht.** Gemessen wird die Zeichnung selbst, ohne Breitenbezug. Was sich auf `100%` setzt, misst für jedes Bild dasselbe und fällt aus der Prüfung – zu Recht, denn so ein Bild hat seinen festen Rahmen schon.
>
> **Was er meldet, wo nichts wandert.** Eine Zeichnung, die nur nach rechts und unten wächst, bewegt ihre Tinte nicht, misst sich aber verschieden. Dafür steht `steady: false` an der Szene.

11.9.9 Folien ohne Titel

Ein nacktes `==` lässt den Titelbalken weg; der Rumpf beginnt oben und bekommt die Höhe, die sonst der Balken belegt hätte. Das ist die Folienart für die eine große Formel – und das Ziel eines Morphs, der in die Mitte fliegen soll:

```
==
#place(center + horizon, morph(<ableitung>, text(size: 2.4em)[
  $f'(x) = lim_(h -> 0) (f(x+h) - f(x)) / h$
]))
```

In der Argumentform sind alle drei Schreibweisen erlaubt: `slide[Rumpf]` ohne Titel, `slide(none)[Rumpf]` ausdrücklich ohne, `slide([Titel])[Rumpf]` mit.

11.10 Labels: jede gebaute Form ansprechen

Jede Form, die typstage selbst zeichnet – Grundfläche, Kopfband, Folientitel, Fußzeile, Fortschritt, Kasten, Merksatz, große Aussage, Titel- und Abschnittsfolie, Ersatzfläche eines Videos –, trägt ein festes Typst-Label. Eine gewöhnliche `show`-Regel genügt dann, kein Theme-Schlüssel, kein Fork.

```
#import "@preview/typstage:0.1.0": *

#show label("ts-slide-header-band"): set rect(fill: rgb("#4c1d95"))
#show label("ts-slide-title"): set text(fill: rgb("#fde047"), style: "italic")
#show label("ts-card"): set block(fill: rgb("#eef2ff"))
#show label("ts-statement"): set text(fill: rgb("#be123c"), weight: "bold")

#show: presentation.with(theme: themes.default)
```

Die **Flächen** nehmen `set rect(..)`, `set block(..)`, `set circle(..)` oder `set line(..)`; die **Schriften** nehmen `set text(..)`. Beides wirkt zur Übersetzungszeit und steht deshalb gleich in HTML und PDF – ausgenommen die sechs Labels unter /Medien und Handout/, die im Browser dem echten `<video>` oder `<iframe>` weichen.

> ACHTUNG
> **Bei den Flächen** wirkt die Kurzform, die Langform nicht:
>
> ```
> #show label("ts-slide-progress"): set rect(fill: green)          // ja
> #show label("ts-slide-progress"): it => { set rect(fill: green); it }   // nein
> ```
>
> Die Kurzform legt die Stilregel **um** das gefundene Element, die Langform **hinein** – und im Rechteck steckt kein zweites Rechteck. Bei den 16 Schrift-Labels sind beide Schreibweisen gleichwertig.

11.10.1 Wo die Regel stehen muss

Vor `#show: presentation`. Diese eine Stelle erreicht alles: den Folienhintergrund, Kopf, Fuß und Fortschritt, die Titelfolie und jedes bewegte Element.

`style` erreicht das nicht: der Haken liegt um den **Folienrumpf**, und Kopf, Fuß, Fortschritt sowie Titel- und Abschnittsfolie entstehen daneben. Gemessen, jede der 38 Regeln einzeln: aus `style` heraus wirken genau die 13, die im Folienrumpf stehen – `ts-card…`, `ts-callout…`, `ts-statement` und die drei `ts-media-…`. Die übrigen 25 bleiben dort stumm, ohne Warnung.

> **ACHTUNG**
> Eine `show`-Regel **hinter** `#show: presentation` erreicht ein getracktes Element (`anim`, `morph`) nicht:
>
> ```
> #show: presentation.with(theme: themes.default)
> #show label("ts-statement"): set text(fill: green)   // zu spät
> == Eine Folie
> #statement[fest]
> #anim(statement[bewegt])
> ```
>
> Hier ist `fest` grün und `bewegt` schwarz; eine Zeile weiter oben sehen beide gleich aus. Im PDF fällt es nicht auf, weil dort nichts zweimal gesetzt wird. Das gilt für jede `#show`-Regel, nicht nur für Label-Regeln.

11.10.2 Was eine Label-Regel ändert und was nicht

Erreichbar ist, was das Paket **nicht** als ausdrückliches Argument schreibt. Für die Schrift ist das alles; für die Flächen sind es `fill` und `stroke` überall und `radius` dort, wo die Form eine Rundung hat.

`width` steht überall als Argument und ist deshalb nirgends erreichbar. Bei `height` gibt es drei Ausnahmen: `ts-card`, `ts-card-bar` und `ts-callout` bekommen ihre Höhe als `auto`, und `auto` schlägt keine Regel.

```
#show label("ts-card"): set block(height: 150pt)   // wirkt
#show label("ts-card"): set block(width: 30%)      // wirkt nicht
```

Bei den Chrome-Flächen, den Grundflächen und dem Handout-Rahmen wirkt weder das eine noch das andere. Nicht erreichbar ist auch die **Anordnung** der Folie: Kopfhöhe, Abstand der Titellinie, Sitz des Balkens entstehen in `place` und `layout`. Dafür sind die Theme-Schlüssel da.

> **ACHTUNG**
> Eine Regel auf `block` oder `rect` reicht nach **innen**: Sie gilt für die gelabelte Fläche und für jeden Block darin. Bei `fill`, `stroke` und `radius` ist das abgefangen, bei den Abständen nicht – dann verschiebt eine Label-Regel die Folie:
>
> ```
> #show label("ts-card"): set block(below: 60pt)
> == Eine Folie
> #card(title: [Kasten])[Rumpf]
> #callout(title: [Merke])[Merksatz]
> ```
>
> Der Merksatz rückt nach unten und alles unter ihm mit, um `60pt` minus dem Blockabstand, **je Kante**. Labels sind für Schrift und Fläche gedacht; wer Abstände will, nimmt die Argumente der Bausteine oder die Theme-Schlüssel.

11.10.3 Das vollständige Verzeichnis

Die Namen folgen einem Schema: `ts-`, dann der **Ort**, dann der **Teil**. Steht `slide` **vorn**, geht es um die gewöhnliche Folie; steht es hinter `title` oder `section`, um jene Folienart – `ts-slide-title` und `ts-title-slide-title` sind zwei verschiedene Dinge, und ein Fehlgriff bleibt stumm.

Ein Label, das dieses Theme gerade nicht zeichnet – ein Kopfband bei `header: "run"` etwa –, gibt es auf dieser Folie nicht, und eine Regel darauf tut nichts.

Die gewöhnliche Folie

Label	Was es ist	Regel
`ts-slide-ground`	die Grundfläche	`rect`
`ts-slide-header-band`	das Kopfband, nur bei `header: "band"`	`rect`
`ts-slide-header-text`	die laufende Kopfzeile, nur bei `header: "run"`	`text`
`ts-slide-header-rule`	die Haarlinie darunter, nur bei `header: "run"`	`rect`
`ts-slide-title`	der Folientitel, bei allen drei Kopfarten	`text`
`ts-slide-title-rule`	die Linie darunter, nur bei `rule-size > 0pt`	`rect`
`ts-slide-footer`	die Fußzeile	`text`
`ts-slide-number`	die Foliennummer darin	`text`
`ts-slide-footer-rule`	die Haarlinie darüber, nur bei `footer-rule > 0pt`	`rect`
`ts-slide-progress`	der Fortschrittsbalken, bei `progress: "tick"` der wandernde Reiter	`rect`
`ts-slide-progress-track`	seine Bahn, nur bei `progress: "tick"`	`rect`

Die Titelfolie

Label	Was es ist	Regel
`ts-title-slide-ground`	ihre Grundfläche	`rect`
`ts-title-slide-band`	das Band oben, nur in `themes.lesson`	`rect`
`ts-title-slide-title`	ihr Titel	`text`
`ts-title-slide-subtitle`	ihr Untertitel	`text`
`ts-title-slide-rule`	die Zierlinie; `themes.editorial` hat zwei, `themes.plain` keine	`rect`
`ts-title-slide-byline`	die Zeile aus Verfasser und Datum	`text`

Die Abschnittsfolie (einen Untertitel hat sie nicht)

Label	Was es ist	Regel
`ts-section-slide-ground`	ihre Grundfläche	`rect`
`ts-section-slide-bar`	der Balken links, nur in `themes.lesson`	`rect`
`ts-section-slide-title`	ihr Titel	`text`
`ts-section-slide-rule`	die Zierlinie; `themes.night` hat zwei, `themes.lesson` keine	`rect`
`ts-section-slide-parent`	die Zeile darüber mit den übergeordneten Abschnitten. Erst ab der zweiten Gliederungsebene	`text`

Die Bausteine im Folienrumpf

Label	Was es ist	Regel
`ts-card`	der Kasten: Fläche, Rand, Rundung und alles darin	`block`
`ts-card-bar`	der farbige Reiter darüber, nur bei `box: "bar"`	`block`
`ts-card-title`	seine Überschrift	`text`
`ts-card-disc`	die Scheibe der Nummer, nur bei `number:`	`circle`
`ts-card-number`	die Ziffer darin	`text`

Label	Was es ist	Regel
`ts-card-body`	sein Rumpf	`text`
`ts-callout`	der Merksatz. Der Balken links ist kein eigenes Label, er ist der linke `stroke` dieses hier	`block`
`ts-callout-title`	seine Überschrift	`text`
`ts-callout-body`	sein Rumpf	`text`
`ts-statement`	die große Aussage. `size` wirkt als Faktor darauf, weil `statement` in `em` misst	`text`

Medien und Handout

Label	Was es ist	Regel
`ts-media-fallback`	die Ersatzfläche, die im PDF für ein bewegtes Element steht. Nur eine Hülle: `radius` sieht man nicht, `fill` schon	`block`
`ts-media-fallback-empty`	der graue Kasten darin, wenn kein `fallback:` angegeben ist	`block`
`ts-media-poster`	die graue Fläche eines `video` ohne `poster:`	`rect`
`ts-handout-frame`	der gerahmte Kasten einer Folie auf der Handout-Seite	`block`
`ts-handout-lines`	die Schreiblinien daneben oder darunter	`line`
`ts-handout-note`	die Sprechernotiz, wo es eine gibt	`text`

> **HINWEIS**
> **Ein Theme mit eigener Titelfolie zeichnet keins dieser Labels.** `title-slide` und `section` sind Funktionen und malen ihr Bild selbst; wer eine eigene mitbringt, verliert die sechs beziehungsweise vier Labels dieser Folienart, und nichts warnt davor.
>
> **Typst-Labels und die CSS-Klassen der Laufzeit sind zwei Namensräume.** `.ts-slide` im Stylesheet ist der `<section>` einer Folie im Browser, `ts-slide-title` ein Typst-Label. Typsts HTML-Ausgabe legt an manche Formen ein `data-typst-label`-Attribut; das ist Beiwerk von Typst, kein Versprechen dieses Pakets.

11.11 `info()`: was das Deck über sich selbst weiß

Labels sagen, wie eine gebaute Form aussieht, nicht was in ihr steht. Die Foliennummer, der Bruch, der Kapitelname in der Kopfzeile: `info()` gibt sie heraus.

```
#context {
  let deck = info()
  [#deck.section.title #h(1fr) #deck.slide.number / #deck.slide.total]
}
```

Es ist dieselbe Lesung, die die eingebaute Fußzeile macht; beide können deshalb keine verschiedenen Zahlen drucken.

Feld	Was darin steht
`title`, `subtitle`	Titel und Untertitel des Decks
`author`, `date`	Ebendaher. `date` ist ein `datetime` oder Inhalt

Feld	Was darin steht
`slide.number`	Diese Folie. Gezählt wie die Fußzeile zählt, Titel- und Abschnittsfolien also nicht mit
`slide.total`	So viele Folien werden gezählt
`slide.numbered`	Ob diese Folie mitgezählt wird. Auf einer Titel- oder Abschnittsfolie `false`
`step.number`	Der Schritt, auf dem der aufrufende Inhalt selbst steht
`step.total`	So viele Schritte hat diese Folie
`section.number`	Der wievielte Abschnitt gerade läuft, `0` vor dem ersten
`section.total`	So viele Abschnitte hat das Deck
`section.title`	Sein Titel, oder `none` vor dem ersten
`levels`	Ein Eintrag je Struktur-Ebene, von außen nach innen. Leer bei `slide-level: 1`
`outline`	Die ganze Gliederung, ein Eintrag je Abschnittsfolie

`section` meint immer die Ebene direkt über der Folie; bei der Vorgabe `slide-level: 2` ist das die einzige. Wer mehr Ebenen hat, findet sie in `levels` und `outline`. Ein Eintrag beider trägt `depth`, `title` und `number`; `number` zählt die Abschnitte dieser Ebene im **ganzen** Deck durch und geht nie zurück. Der Vergleich von `outline.at(j).number` mit `levels.at(..).number` sagt damit, ob ein Eintrag vorbei ist, läuft oder noch kommt; die weiteren Felder stehen in der API-Referenz.

```
#context {
  let d = info()
  stack(spacing: 0.6em, ..d.outline.map(e => {
    let lauf = d.levels.at(e.depth - 1).number
    text(
      weight: if e.number == lauf { "bold" } else { "regular" },
      fill: if e.number <= lauf { black } else { luma(60%) },
      [#h((e.depth - 1) * 1.4em)#e.title],
    )
  }))
}
```

Eine Zahl steht bewusst daneben: Sprecheransicht und Übersicht zählen **alle** Folien, `info().slide.total` zählt wie die Fußzeile und lässt Titel- und Abschnittsfolien aus.

11.11.1 Zwei Zählungen, nicht eine

Das Paket zählt in Folien **und** in Schritten: eine Folie ist ein Bild, ein Schritt ist ein Tastendruck.

`step.number` ist der Schritt, auf dem der aufrufende Inhalt selbst steht: im Rumpf einer Folie `1`, innerhalb eines `anim`, `stagger` oder `alternatives` der Schritt jener Einblendung. Eine Anzeige, die den laufenden Schritt nennt, muss deshalb **in** den Einblendungen sitzen, denn der Browser setzt nichts neu:

```
#let stand = context {
  let d = info()
  [Schritt #d.step.number von #d.step.total]
}

== Vier Fassungen
#alternatives(stand, stand, stand, stand)
```

Auf dem Papier gibt es keinen laufenden Schritt: die Seite zeigt die Folie im Endzustand, und `step.number` ist dort gleich `step.total`.

11.11.2 Wohin die eigene Fußzeile gehört

Auf einer Titel- oder Abschnittsfolie zeichnet typstage keine Fußzeile, und in den Zahlenplatz gehört dort nichts. `slide.numbered` sagt, wann das der Fall ist:

```
#let fusszeile = context {
  let d = info()
  let zahl = if d.slide.numbered [#d.slide.number / #d.slide.total] else []
  place(bottom + right, text(size: 12pt, fill: muted, zahl))
}
```

Auf einer gewöhnlichen Folie steht sie im Rumpf:

```
== Eine Folie
#fusszeile
Der Text der Folie.
```

Auf Titel- und Abschnittsfolien muss sie ins Theme: beide Bilder sind Funktionen, und eine Funktion, die eine andere umschließt, ergänzt sie.

```
#let basis = themes.default
#let mit(f) = (t, s, geo) => { f(t, s, geo); fusszeile }

#show: presentation.with(
  theme: basis + (title-slide: mit(basis.title-slide), section: mit(basis.section)),
)
```

> **ACHTUNG**
> **Nicht über `style:`.** `style: it => { fusszeile; it }` sieht nach der bequemen Abkürzung aus. Der Haken ist aber zugleich die Vorlage, mit der jedes bewegte Element ein zweites Mal gesetzt wird: alles, was dort **zeichnet**, wird in jedem Sprite mitgezeichnet, und die Fußzeile steht dann mehrfach auf der Folie. Im Rumpf steht sie einmal.
>
> Eine im Rumpf platzierte Fußzeile sitzt am unteren Rand des **Rumpfes**, nicht der Folie; dazwischen liegt der `foot-gap` des Themes. Ein `dy:` am `place` schiebt sie dorthin, wo sie hin soll.

> **ACHTUNG**
> `info()` liest den Stand der Folie, die gerade gesetzt wird, und braucht deshalb ein `context` um sich. **Vor** der Präsentation bricht es mit einer Meldung ab, statt Nullen zu liefern. **Danach** nicht: wer die Folien als Argumente übergibt und unter den Aufruf noch ein `info()` schreibt, bekommt weiter die Zahlen der letzten Folie.

11.11.3 `deck-outline()`: wie das Deck geschnitten ist

`info()` sagt, **wo** man steht, nicht wie das Ganze gegliedert ist. Wer sich eine Navigationsleiste baut, braucht genau das. `deck-outline()` gibt einen Eintrag je Abschnitt heraus, in ihrer Reihenfolge:

```
#context for a in deck-outline() [
  - #a.number. #a.title -- Folien #a.first bis #a.last (#a.count)
]
```

`first`, `last` und `count` zählen **transitiv**: unter einen Abschnitt der Tiefe 1 fallen auch die Folien seiner Unterabschnitte. Ein Abschnitt ohne Folien hat `none` bei `first` und `last` und `0` bei `count`.

Gezählt werden nur Überschriften auf Dokumentebene, also die **zwischen** den Folien. Eine Überschrift **in** einer Folie – `slide(none)[= Jede Karte lügt]` – ist ein Folientitel und eröffnet keinen Abschnitt. Wer eine Navigationsleiste will, setzt die `=` also zwischen die Folien; `examples/gliedern.typ` macht das vor.

ACHTUNG
Ein fremdes Paket, das die Gliederung über `query(heading)` sucht, findet nichts: die Überschriftennotation zerlegt den Rumpf an seinen Überschriften und kopiert `depth` und `body` heraus, das Element selbst fällt weg. Das gilt in **beiden** Ausgaben. `deck-outline()` ist die Antwort darauf.

12 Weitergeben

Das Ziel dieses Kapitels: den Vortrag dorthin bringen, wo er gehalten wird.

12.1 Wo die Datei liegen kann

Die HTML-Datei ist statisch. Was Dateien ausliefert, liefert auch sie aus: GitHub Pages, ein Webplatz der Hochschule, ein S3-Eimer. Aus `file://` geöffnet verhält sie sich wie eine vom Server, samt Sprecheransicht.

Medien liegen aber neben der Datei, nicht darin. Fehlt die `clip.mp4` nach dem Hochladen, bleibt das Videofenster leer – auch wenn das Deck lokal noch lief.

13 Was es nicht kann

Die Grenzen, gesammelt an einer Stelle statt verstreut über die Kapitel.

13.1 Barrierefreiheit

Das ist die härteste Grenze des Pakets. Die Folien sind SVG-Umrisse, Text darin ist als Pfad gezeichnet. Nichts im Browser ist auswählbar, durchsuchbar oder von einem Bildschirmleser lesbar; es gibt keine Textalternative und keine Lesereihenfolge.

Was hingegen geht: Das Dokument trägt aus `text.lang` ein `lang`-Attribut. Die Navigation ist vollständig über die Tastatur bedienbar, und `?` zeigt die ganze Tastenliste. Farbe und Kontrast gehören dem Theme und damit dir; `themes.plain` ist das dunkelste der fünf Themes auf Weiß. Wer sein System auf weniger Bewegung eingestellt hat, bekommt ein Deck, das darauf reagiert (`prefers-reduced-motion`); `transition: "none"` und `enter: "none"` setzen dasselbe für alle durch.

> **ACHTUNG**
> Sitzt jemand im Raum, der einen Bildschirmleser nutzt: Die ehrliche Lösung ist, zusätzlich die PDF auszugeben und auf jeder Folie vorzulesen, was dort steht. Die PDF aus derselben Quelle trägt echten Text.

13.2 Eine Grenze bei verfolgten Elementen

Sehr dicke Striche. Eine Linie misst 0pt hoch; ihre Farbe liegt außerhalb des Kastens. Damit ein flächenloses Element überhaupt erscheint, bekommt es eine Schrifthöhe Luft nach jeder Seite. Was dicker aufträgt, wird beschnitten. Solche Striche gehören in einen `block` oder `rect` mit eigener Höhe, dann gilt dessen Maß.

> **HINWEIS**
> Sonst braucht die Breite eines verfolgten Elements keine Aufmerksamkeit: Das Paket sieht dem Inhalt an, ob er den angebotenen Platz ausfüllen will. Ein `align(center, …)` in einem `anim` zentriert wie im PDF, und ein verfolgtes Element in einer `auto`-Rasterspalte lässt die `1fr`-Nachbarspalte stehen.

Was gar keine Fläche hat. Ein senkrechter Strich misst null breit, ein `place` null in beiden Richtungen. Beide sind versorgt: Der Strich bekommt seine Luft wie jedes flächenlose Element, und beim `place` tauschen Marke und Inhalt die Reihenfolge, damit im Fluss kein Platz belegt wird:

```
#anim(at: 2, place(top + left, dx: 20pt, dy: 50pt,
                   rect(width: 20pt, height: 20pt)))
```

Und wenn doch keines von beidem greift, sagt es die Laufzeit, statt das Element zu verlieren. Zwei Fälle bleiben: eine Marke, die null breit oder null hoch misst, und eine, die tiefer als vier verfolgte Elemente ineinander liegt. Beide gehen einmal je Element in die Konsole des Browsers, mit dem Ausweg dazu – das Element in einen Kasten mit Größe setzen oder ihm eine Breite geben.

Gemeldet wird das erst im Browser: Beim Übersetzen weiß Typst nicht, ob ein Inhalt eine Fläche hat, erst dort liegt das Rechteck da und lässt sich messen.

13.3 Reichweite

Geprüft in Chrome, Firefox und Safari auf macOS und auf einem iPhone. Nicht geprüft: ältere Browser, Windows, Android. Die Laufzeit benutzt die Web Animations API, `ResizeObserver`, `PointerEvent` und CSS `zoom`; ein Browser von vor etwa 2023 wird also irgendwo zu kurz greifen.

13.4 Größe und Tempo

Eine Folie wird so oft gesetzt, wie sie Zustände hat, und jedes verfolgte Element noch einmal in einem eigenen Rahmen. Die Übersetzungszeit wächst also mit den Schritten, nicht mit den Folien, und `flipbook` wächst mit den Bildern. Für ein gewöhnliches Deck bleibt das im Sekundenbereich; wer hundert Folien mit je einem Daumenkino plant, misst besser, statt zu schätzen.

14 Wenn nichts passiert

Die Stolpersteine, ungefähr in der Reihenfolge, in der man über sie fällt.

Keine HTML-Ausgabe — `--features html` fehlt.

Das Deck besteht nur aus der Titelfolie — die beiden Schreibweisen sind vermischt. Ein `slide(...)` im Rumpf einer Show-Regel ergibt keine Folie und auch keinen Fehler. Entweder `= …` und `== …` schreiben, oder `slide(...)` an `presentation` übergeben.

Der erste Absatz fehlt — Inhalt vor der ersten Überschrift gehört zu keiner Folie. Text bricht die Übersetzung ab, ein Bild geht wortlos verloren.

Die Folientitel übergehen ein `#set heading` — es steht nach der Show-Regel und damit außerhalb von deren Geltungsbereich. `style:` erreicht sie trotzdem.

`#pause` tut nichts — es steht in einer Rasterzelle oder Tabelle, und dort ist nichts zu zerlegen. `anim` geht überall dorthin, wo Inhalt hingeht.

Ein Übergang oder eine Wirkung wird beim Namen abgewiesen — den Namen gibt es nicht. Die Meldung nennt alle, die es gibt.

Die Stichpunkte neben einem Applet fangen bei Schritt drei an — ein `embed` verbraucht keinen Schritt, etwas davor aber schon. Gezählt werden die Aufdeckungen, nicht die Elemente.

Eine fliegende Formel hat die falsche Schrift — ein verfolgtes Element wird in einem eigenen Rahmen gesetzt, den ein `#set` nicht erreicht. `style:` an `presentation` erreicht beides.

Ein eingebetteter Rahmen bleibt leer und bekommt keine Aufträge — das Dokument darin hat sich nicht mit `postMessage({typstage: 1, ready: 1})` gemeldet.

Ein Applet-Rahmen bleibt leer — das Applet wird von `geogebra.org` geladen, ohne Netz gibt es nichts zu laden. `codebase` zeigt auf eine örtliche Kopie.

Ein `ggb-run`-Befehl bleibt wirkungslos — er ist einer von GeoGebras Skriptbefehlen, die `evalCommand` nicht annimmt. `ggb-set`, `ggb-style`, `ggb-show` und `ggb-hide` nutzen die Schnittstelle, die das kann.

Der Bau bricht ab und nennt zwei Applets — zwei Rahmen auf einer Folie und kein `target`. Hier wird mit Absicht nichts geraten.

Die Farben des Applets ändern sich nach dem Zurückblättern — GeoGebra vergibt beim Neuaufbau die nächste Farbe seiner Palette. Die Farbe lässt sich auf `"1-"` festlegen.

Ein Kreis im Applet ist eine Ellipse — x-Bereich und y-Bereich von `ggb-view` passen nicht zur Form des Kastens.

Ein Tween läuft nicht — er steht auf Schritt 1, wo die Laufzeit ihn auf seinen Zielwert setzt statt ihn zu fahren. Oder er hat einen Bereich statt einer Schrittnummer bekommen.

Zwei Schieber liegen übereinander — `position` zählt bei einem mit `Slider` gebauten Schieber in Pixeln, nicht in Koordinaten.

Ein Punkt lässt sich in der Sprecheransicht nicht ziehen — er ist mit `Point(k, 0.3)` gebaut und an diesen Parameter geheftet.

Ein Rahmen ist auf dem Projektor winzig, auf dem Laptop richtig — sein Inhalt ist in Pixeln bemessen statt in `em`.

„constructing a document is only supported in the bundle target“ — die Datei benutzt `bundle` und braucht deshalb `--format bundle`.

Die Sprecheransicht öffnet nicht — `window.open` braucht einen echten Tastendruck.

15 API-Referenz

Erzeugt aus den Kommentaren der Quelldateien. Die Reihenfolge folgt dem Aufbau des Pakets: erst die Präsentation und ihre Folien, dann die Bausteine, dann Medien und Brücke, zuletzt die Maße und Farben.

15.1 Die Präsentation

15.1.1 `bundle`

```
bundle(
  body,
  html: "talk.html",
  slides: "slides.pdf",
  handout: none,
  per-sheet: 3,
  ..args,
)
```

All outputs in one run.

Since 0.15 Typst can write several files from one compilation. That fits this package, since everything sits in one source anyway: the talk, the slide deck and the handout differ only in target and in one setting. Instead of compiling three times, once:

```
typst compile --features bundle,html --format bundle talk.typ output
```

```
#bundle(
  theme: themes.lesson,
  title: [Completing the Square],
  handout: "handout.pdf",
)[
  = A section
  == A slide
  Text.
]
```

`html`, `slides` and `handout` are file names; `none` leaves out that output. `per-sheet` is the number of slides per handout page. Everything else goes to `presentation` unchanged.

Two things worth knowing. The bundle is explicitly experimental in Typst. And a file that uses `bundle` can **only** be compiled with `--format bundle`: `typst compile talk.typ talk.pdf` aborts with „constructing a document is only supported in the bundle target“. Anyone who wants both writes the body into a `#let` and calls `presentation` by hand.

Verified: the counters start over for each output. The slide deck numbers its slides 1, 2, 3 and does not continue counting where the HTML version left off, even though Typst runs introspection across the whole bundle.

Name	Vorgabe
`body`	–
`html`	`"talk.html"`
`slides`	`"slides.pdf"`
`handout`	`none`

Name	Vorgabe
`per-sheet`	`3`
`..args`	–

15.1.2 `presentation`

```
presentation(
  ..slides,
  title: none,
  subtitle: [],
  author: [],
  date: none,
  assets: "inline",
  theme: themes.default,
  palette: (:),
  transition: "slide",
  speaker-view: (:),
  transition-duration: 420,
  duration: 520,
  style: it => it,
  width: auto,
  height: auto,
  margin: auto,
  handout: false,
  overflow: "none",
  drift: "error",
  slide-level: 2,
)
```

Build the deck.

Two notations, the same output. Either the slides as arguments:

```
#presentation(title-slide(title: [Title]), section[Part], slide([First])[…])
```

… or as a show rule, and then the headings separate the slides:

```
#show: presentation.with(title: [Title], transition: "slide")
= A section
== A slide
Content …
```

Two targets, one source:

```
typst compile deck.typ deck.html --format html --features html
typst compile deck.typ deck.pdf
```

`slide-level:` is where the deck is cut. A heading **above** it becomes a section slide, a heading at it or below it becomes a slide. The default is 2, and that is the rule this package always had: `=` opens a section, `==` a slide.

```
#show: presentation.with(title: [Analysis], slide-level: 3)
= Part I
== Sequences
=== What a sequence is
A map from the naturals.
```

`= Part I` and `== Sequences` each become a section slide, `===` becomes the slide. Both transition slides come for free: a section heading **is** the transition slide here, so there is nothing to switch on and no hook to write. `slide-level: 1` makes every heading a slide and leaves the deck without any structure level.

A deeper level is drawn more quietly by all five bundled themes, smaller and with the titles it hangs under set above it. A theme of its own reads `s.depth` and `s.parents` off the section record and may ignore both, and then every level looks alike.

What the deck knows about its structure is in `info()`: `section` is unchanged and always means the level directly above the slide, `levels` has one entry per structure level, and `outline` is the whole thing.

`theme:` determines the whole look: colors, typeface, title bar, footer, progress, title and section slide. Bundled are `themes.default` (the default), `themes.lesson`, `themes.night`, `themes.plain` and `themes.editorial`; each of them can be varied with `themes.night + (accent: blue)`. The `style` hook stays untouched by this and sits further **inside**: whatever is set there overrides the theme.

`palette:` changes the colors and leaves the design alone. It is a dictionary over the eight color entries and it overwrites **partially**, so `palette: (accent: blue)` moves the accent and nothing else. Five are bundled, `palettes.light`, `palettes.mono`, `palettes.textbook`, `palettes.parchment` and `palettes.dark`, and each of them composes with each theme:

```
#show: presentation.with(theme: themes.lesson, palette: palettes.dark)
```

Two colors of a theme are not palette entries: `title-fill` and `rule-fill`. All five bundled themes let them follow, either as a function of the palette or as `none`, which means the accent and follows with it. A theme of your own that names a fixed color there keeps it under every palette, which is deliberate.

Both changed type with this: reading `themes.X.title-fill` used to give a color and now gives a function, and `rule-fill` gives `none` where it gave the accent. Writing them, `themes.X + (title-fill: red)`, is unchanged.

`speaker-view` says what the presenter view shows. Everything is on unless switched off, so a deck that says nothing gets the whole thing:

```
#show: presentation.with(speaker-view: (
  clock: false,                       // no class clock
  target: false,                      // no planned length
  pen: (colors: (red, green, blue)),  // the drawing bar's colours
))
```

A tile that is switched off takes its keys with it: with `clock: false`, `t` and `⇧t` do nothing and no longer stand in the key bar. A view that advertises a key which does nothing is worse than one that is missing it. `tools: false` removes the drawing bar the same way.

The PDF is a handout: one page per slide, every tracked element in its final state. What belongs only to the motion, the notes, the slide transitions, the bridge jobs, are state updates without output and fall away by themselves.

`overflow` is a checking pass over the deck, off by default. It measures every slide body against the room the theme gives it and names the ones that do not fit, with the earliest step on which the overrun can be on the screen. Title and section slides are not measured: the theme draws them with `place` and they have no body block.

- `"none"`: nothing is measured. The default.
- `"error"`: the whole deck is built, and it then stops with **every** place at once rather than the first.
- `"record"`: it carries on and files a record per finding instead, for a tool to read. The deck has to be on `"record"` for this; on `"error"` the command below stops with the error too:

```
typst eval --target html --features html --in deck.typ \
  'query(<typstage-overflow>).map(e => e.value)'
```

The same setting can be raised from the command line, so a build script can measure a deck without editing it:

```
typst compile --features html --format html \
  --input typstage-overflow=error deck.typ deck.html
```

The input raises, it never lowers. Of the two the stricter one wins, `"none"` < `"record"` < `"error"`, so a run cannot switch off a check the deck asked for.

It is not meant to stay on while writing. Measured over the six example decks: in HTML it costs noticeably more time, between 1.2 and 1.5 times depending on the deck and on how the process start is accounted for. On paper it costs a few milliseconds per deck, small but repeatable: there the check runs without the step arithmetic.

Why a deck of slides needs this more than a document does: a slide goes into an SVG frame of fixed size and is scaled in the browser, so what sticks out is cut away or drawn beside the slide. A page one leafs through shows an overrun; a talk one clicks through shows it at the projector.

`drift` is the second check, and unlike `overflow` it is on. Every `scene` measures its frames, and a scene whose frames come out different sizes is named: a CeTZ canvas is as large as what it holds, so a wider frame puts the drawing somewhere else inside its box and paging through it the whole picture travels while only one point should move.

- `"error"`, the default: the deck is built and then stops with every scene at once. `scene(steady: false)` says the frames of that one scene are meant to differ and takes it out of the check.
- `"record"`: it carries on and leaves a record per finding, for a tool to read, the same way `overflow: "record"` does:

```
typst eval --target html --features html --in deck.typ \
  'query(<typstage-drift>).map(e => e.value)'
```

- `"none"`: the frames are not measured at all.

On by default where `overflow` is not, and for two reasons. Only decks that use `scene` pay for it at all – measured on a scene of 28 CeTZ frames, 434 ms without and 536 ms with, so about 100 ms for that scene – where `overflow` measures every slide of every deck and costs 1.2 to 1.5 times the whole compilation. And what it finds is invisible while writing: every frame on its own looks right, and only paging through shows the drawing travelling. Only the browser branch measures. On paper a scene is one still image, and a still image does not travel.

Name	Vorgabe
`..slides`	–
`title`	`none`
`subtitle`	`[]`
`author`	`[]`
`date`	`none`
`assets`	`"inline"`
`theme`	`themes.default`
`palette`	`(:)`
`transition`	`"slide"`
`speaker-view`	`(:)`

Name	Vorgabe
`transition-duration`	`420`
`duration`	`520`
`style`	`it => it`
`width`	`auto`
`height`	`auto`
`margin`	`auto`
`handout`	`false`
`overflow`	`"none"`
`drift`	`"error"`
`slide-level`	`2`

15.2 Folien

15.2.1 `class-clock`

```
class-clock(minutes)
```

Plans a class clock for this slide: how many minutes the work on it is meant to take.

It starts nothing. `Shift+T` in the presenter view offers the number, the speaker confirms or changes it, and only then does the clock run – the deck knows how long the task was meant to take, the room decides how long it actually gets. A slide carries at most one; a second call replaces the first, like a second `speaker-note`.

15.2.2 `deck-outline`

```
deck-outline()
```

How the deck is cut, section by section.

`info()` says where you are; this says how the whole thing is divided. One entry per section, in the order they come, each with the slides beneath it:

```
#context for a in deck-outline() {
  [#a.number. #a.title -- slides #a.first to #a.last (#a.count)]
}
```

`first`, `last` and `count` are transitive: a depth-1 section counts the slides of its sub-sections too. A section with no slides under it has `none` for `first` and `last`, and 0 for `count`.

Read straight off the state every slide already carries. No `query`, no second walk over the document, and the same answer in both outputs.

15.2.3 `info`

```
info()
```

What the deck knows about itself, read from inside a slide.

```
#context {
  let deck = info()
  [#deck.section.title #h(1fr) #deck.slide.number/#deck.slide.total]
}
```

It is the same reading the built-in chrome does. Every number the package prints on a slide, the footer, the fraction, the length of the progress bar and the running header, comes out of this function and out of no second count, so a hand-built footer and the built-in one cannot disagree.

What comes back, as a dictionary:

- `title`, `subtitle`, `author`, `date`: the deck's own particulars, as `presentation` or a `title-slide` received them.
- `slide.number`, `slide.total`: this slide and how many there are. Counted the way the footer counts, so title and section slides are not in it.
- `slide.numbered`: whether this slide is one of the counted ones. It is `false` on a title slide and on a section slide, and `number` then holds the last slide counted before it, 0 on a cover that opens the deck. A footer can therefore leave its counter slot clear instead of printing a zero into it.
- `step.number`, `step.total`: this deck counts in steps as well as in slides, which no footer can guess at. `number` is the step the calling content itself stands on: 1 in the body of a slide, and inside an `anim`, a `stagger` or an `alternatives` the step of that reveal, its first one where it covers several. On paper a slide is one page in its final state, so `number` is `total` there.
- `section.number`, `section.total`, `section.title`: which section the slide belongs to, how many the deck has, and its title. The section is always the level directly above the slide, so at the default `slide-level: 2` this is the `=` heading and nothing about it has changed. Before the first such heading, `number` is `0` and `title` is `none`. A deck at `slide-level: 1` has no structure level at all, and then all three read `0`, `0` and `none`.
- `levels`: one entry per structure level, from the outermost inwards, so `levels.last()` is the same section as above and `levels.first()` the outermost part. Empty at `slide-level: 1`. Each entry carries `depth` (1 for `=`, 2 for `==`), `title` (`none` while no section of that level is running), `number` and `total` (counted across the whole deck, the way `section` counts), and `index` and `count` (counted among the siblings under the same parent, which is what Beamer prints as `1.2`). `number` never goes back, so it also reads as progress; `title`, `index` and `count` clear when the level above them moves on.
- `outline`: the whole structure of the deck, one entry per section slide in the order they come, each with `depth`, `title`, `number` (the same count as in `levels`) and `here`, which is `true` only on that section slide itself. Comparing an entry's `number` with `levels.at(entry.depth - 1).number` says whether it is past, running or still to come, and that is how a progressive agenda is built.

Only in a context. **Before** any presentation has run there is nothing to read and this stops with a message rather than handing out zeros. **After** one it does not: whoever passes the slides as arguments and writes an `info()` below the call still gets the last slide's numbers. Clearing the deck's own record at the end would close that, and it was measured: a slide carrying one reveal beside a `tiles` went from no layout warning to three „did not converge" ones. A corner nobody stands in is not worth that, and in the show-rule notation nothing comes after the deck anyway.

15.2.4 `section`

```
section(title, depth: 1, transition: none)
```

A section slide.

`depth` is the level in the heading hierarchy the section stands on: `1` for `=`, `2` for `==` and so on, up to one below the deck's `slide-level`. In the heading notation it comes from the heading; here it is written out. The five bundled themes draw a deeper section more quietly, and a theme of your own reads it from `s.depth`.

Name	Vorgabe
`title`	–

Name	Vorgabe
`depth`	`1`
`transition`	`none`

15.2.5 `slide`

```
slide(
  title: none,
  note: none,
  transition: none,
  invert: false,
  ..rest,
)
```

A regular slide.

`title: none`, or a bare `==` in heading form, leaves out the title bar; the body then gets the whole area.

`invert: true` sets this one slide in the palette turned around, for the slide that carries a single number. The ground becomes the palette's text color and the text becomes its ground; `muted`, `border` and `surface` are mixed from those two; `strong` and `accent` carry over unchanged. The chrome follows, so the running header, the footer and the progress bar are set in the same colors as the slide under them.

Only a regular slide inverts. A title slide and a section slide are whole pictures the theme draws itself, and three of the five bundled themes build them from colors an inversion would not reach; neither takes the argument.

Name	Vorgabe
`title`	`none`
`note`	`none`
`transition`	`none`
`invert`	`false`
`..rest`	–

15.2.6 `speaker-note`

```
speaker-note(body)
```

A note for the presenter view, which `n` opens in a second window.

The note has to carry text: the presenter view transports it as a string, so a note made purely of layout would arrive nowhere. That is refused with a message rather than silently dropped.

15.2.7 `title-slide`

```
title-slide(title: [], subtitle: [], author: [], date: none)
```

The title slide.

Name	Vorgabe
`title`	`[]`
`subtitle`	`[]`
`author`	`[]`
`date`	`none`

15.2.8 `transition`

```
transition(kind, ..spec)
```

How this slide comes in, otherwise the presentation's setting applies.

- `"none"`: hard cut.
- `"fade"`: cross-fade, nothing moves.
- `"slide"`, `"push"`, `"cover"`, `"uncover"`: sliding; `from` says where the new slide comes from: `"right"` (default), `"left"`, `"top"`, `"bottom"`. `push` shoves the old one out, `cover` lies down on top, `uncover` pulls the old one away.
- `"zoom"`: `direction: "in"` (default) grows the new one towards you, `"out"` lets it step back from the front.
- `"blur"`: blurred across.
- `"iris"`, `"wipe"`: an aperture. `direction: "open"` (default) opens the new slide out, `"close"` shuts the old one over it. For the wipe, `from` additionally says which edge it starts at.
- `"flip"`, `"cube"`: rotation in space; `axis: "y"` (default) turns about the vertical, `"x"` about the horizontal.

Backwards each one runs as a true reversal. If a morph meets the slide it cross-fades regardless: the movement is then carried by the morph.

Under `prefers-reduced-motion: reduce` every kind but `"none"` becomes the cross-fade, over the same duration. See the manual.

15.2.9 `invert`

The same thing in the heading notation, written into the slide body.

```
== Reached in 2026
#invert
#statement[74 %]
```

A marker, like `#pause`, because a heading carries no arguments. Unlike `#pause` it is only looked for, never split on, so the walk goes all the way down: it is found in a `block`, an `align`, a table cell, a grid, however deeply nested, in the heading itself, and behind `#set` and `#show`. It is not found where the content is handed to a closure – in `context`, `fit`, `anim`, `card` or `alternatives` – and there nothing happens and nothing is said. Measured, those five are the whole of it; `slide(invert: true)` is the form that never depends on the walk.

It prints nothing, so it may stand anywhere in the body; it inverts the whole slide either way, not the part after it.

15.3 Einblenden, Bewegen, Staffeln

15.3.1 `alternatives`

```
alternatives(
  ..variants,
  start: auto,
  align: top + left,
  enter: "fade",
  duration: auto,
  easing: auto,
  inline: false,
  morph: false,
)
```

Several versions of the same thing, each replacing the one before.

```
#alternatives(
  $ (a + b)^2 $,
  $ (a + b)(a + b) $,
  $ a^2 + 2 a b + b^2 $,
)
```

`inline: true` keeps the whole thing in the running line, for versions of a single word or formula.

They all stand in the same place, in a box as large as the largest of them, so nothing around them jumps as they change. Each takes one step; the last one stays for the rest of the slide.

`morph: true` lets the versions fly into one another instead of replacing one another. They all stand in the same place, so the flight is no distance at all and what you see is the glyphs rearranging themselves where they stand – which is what a rewritten formula does:

```
#alternatives(morph: true,
  $ (a + b)^2 $,
  $ (a + b)(a + b) $,
  $ a^2 + 2 a b + b^2 $,
)
```

It works because the versions carry one name and step ranges that do not overlap, and a morph flies from step to step as readily as from slide to slide. A name of your own instead of `true` is allowed and is only needed where the flight has to continue onto the next slide.

A morph has no entrance and no easing curve, so `enter:` and `easing:` are refused rather than quietly dropped. `duration:` is read and is the time of the flight.

On paper only the last one is set, in the same box, so the page keeps the spacing of the slide. Printing all of them would pile them on top of one another.

Name	Vorgabe
`..variants`	–
`start`	`auto`
`align`	`top + left`
`enter`	`"fade"`
`duration`	`auto`
`easing`	`auto`
`inline`	`false`
`morph`	`false`

15.3.2 `anim`

```
anim(
  body,
  at: auto,
  enter: "fade-up",
  exit: "fade",
  after: "hidden",
  duration: auto,
  delay: 0,
  easing: auto,
)
```

Reveal content on particular steps.

`at` is a step selector. `auto`, the default, takes the next free step, so consecutive `anim`s reveal one after another without any numbering. Otherwise: `2` (from step two on), `"1-2"`, `(2, 4)`, `"3"`. An explicit number also moves the cursor along, so a following `auto` carries on after it instead of starting over.

`enter` applies in both directions: paging back plays the same effect in reverse, taking the entrance back. `exit` only concerns a real departure, when an element falls out of its range while moving forward.

`after` says what the element does once its range is behind it, and it has two values.

- `"hidden"`, the default and what an `anim` has always done: it goes, playing `exit`, and keeps the room it had.
- `"dimmed"`: it stays and is drawn muted, so a point remains legible after the talk has moved on. Nothing moves and nothing is recoloured; the element settles to 65 percent opacity, and paging back brings it up again. That number is measured, and the manual says against what.

On paper `after` does nothing at all. A page shows every step at once, and a point that is only quiet because the talk has moved past it has no „past“ on a handout. This is the same rule that already holds for `"hidden"`: what leaves its range in the browser is still printed.

`after` needs a range that ends. `at: auto` and `at: 3` run to the end of the slide, and an element that never leaves has no after; the package says so instead of doing nothing. `at: "3"` is that one step, `at: "2-3"` a range.

Under `prefers-reduced-motion: reduce` every effect keeps its opacity and loses its travel, so `enter` and `exit` become a plain cross-fade of the same length. `after: "dimmed"` is unaffected: it changes opacity and nothing else. See the manual.

`easing` is the curve the element moves on – for the entrance, the departure and the dimming alike. `auto` is the package’s own curve; `"out-back"` overshoots and swings back, `"linear"` arrives at an even pace. A name that does not exist is an error at compile time and not a silent default.

Name	Vorgabe
`body`	–
`at`	`auto`
`enter`	`"fade-up"`
`exit`	`"fade"`
`after`	`"hidden"`
`duration`	`auto`
`delay`	`0`
`easing`	`auto`

15.3.3 `build`

```
build(
  draw,
  steps: 2,
  start: auto,
  enter: "fade",
  duration: auto,
  easing: auto,
)
```

A drawing or a diagram that comes into being step by step.

A CeTZ canvas and a lilaq diagram are one piece, not many: Typst hands out the finished setting, and what was a line and what was a data series in it cannot be reached from outside any more. So there is no `anim` around a part of a drawing. What there is, is the drawing itself, as often as one wants it.

`draw` is called once per step and is handed a question. The examples call it `from`, because it says exactly what `at:` says elsewhere; the name is the deck's own, since it is the lambda's parameter.

- `from(k, value)` gives `value` back once the k-th piece is due, and otherwise the same thing made of air: a colour with alpha 0, a stroke with a transparent brush, a text in `hide`. The piece is therefore never really missing, and every stage measures the same to the point.
- `from(k)` says the same as a boolean, for everything that cannot be recoloured. In CeTZ that is where `hide(…, bounds: true)` belongs.

Whatever carries no number stands there from the start.

```
#build(from => cetz.canvas({
  import cetz.draw: *
  line((0,0), (4,0))                          // there from the start
  line((4,0), (4,3), stroke: from(2, black))  // from step 2
  content((2,3.4), from(3, [hypotenuse]))     // from step 3
}), steps: 3)
```

`steps` is the number of stages and hence the number of steps the drawing takes on the slide. It is said and not guessed: what `draw` does with its question is nobody's business from outside.

`start` is `auto`: the drawing begins on the next free step and pushes the cursor along by `steps`, the way `stagger` and `alternatives` do. A number sets the first step itself.

Exactly one stage is drawn at a time, and that is not a saving but the only arrangement that yields the picture that would stand there if the drawing were set once. Ink adds up: three layers of the same lilaq diagram against one, and 3.7 percent of the pixels differ by more than 8 of 255, the largest deviation 99 – axes, labels and the half-transparent box of the legend get painted three times and grow fatter by it.

On paper only the last stage is set, in a block of the same size: a page shows every step at once, and stacked stages would be overprint. The cursor still runs there, so that `info().step.total` names the same number in both outputs.

Under `prefers-reduced-motion: reduce` nothing changes: the stages fade, they do not travel, and what would fall away is a motion that is not there.

Name	Vorgabe
`draw`	–
`steps`	`2`
`start`	`auto`
`enter`	`"fade"`

Name	Vorgabe
`duration`	`auto`
`easing`	`auto`

15.3.4 `camera`

```
camera(
  target,
  at: auto,
  margin: 16pt,
  duration: 700,
  easing: auto,
)
```

Move in on one detail of the slide, and back out again.

```
#pin(<detail>, card[The measuring head])
…
#camera(<detail>)
```

The camera aims at a `pin`, and at nothing else. That is the package's word for a named piece of a slide, its marker is exactly the rectangle the runtime already measures, and it sits wherever content sits: around a card, around a cell of a table, around one subterm of a formula. Nothing has to be given in coordinates, and nothing has to be counted.

`at` is a step selector, as everywhere else, and the slide is seen through the camera for exactly as long as it is active. That answers the way back out with the notation that is already there: `at: "3"` moves in on step three and back out on step four, `at: "3-5"` holds the crop across three steps, `at: 3` keeps it to the end of the slide.

`auto`, the default, is the next free step **closed**: in on it, out on the one after – and never step one, because step one is the slide as it is entered, and a camera there would mean nobody ever saw the slide whole. That differs from `anim`, where `auto` runs to the end of the slide, and it differs on purpose. An entrance has no natural end – what has appeared stays. A camera move has one: one always comes back out, and coming back out is a keypress like any other, so it is counted like one and shows up in `info().step.total`.

`margin` is how much of the slide stays around the detail, measured in the unzoomed slide. The camera fits the detail plus that margin into the frame; the smaller of the two directions decides, so the whole of it is seen.

A detail that is already as large as the slide gives nothing to travel to, and then the slide stays whole.

Two pins of the same name on one slide are framed together, and the camera shows the box around both.

If two camera moves are active on the same step, the later one in the source wins.

On paper there is no camera. The slide is set whole, exactly as it would be without one – but the steps are counted there too, so the handout's footer names the same number as the talk.

Under `prefers-reduced-motion: reduce` the camera jumps to the crop instead of travelling to it. The package's rule everywhere else too: what stays is the destination, what goes is the travel.

Name	Vorgabe
`target`	–
`at`	`auto`
`margin`	`16pt`
`duration`	`700`

Name	Vorgabe
`easing`	`auto`

15.3.5 `cue`

```
cue(name, ..items, start: auto, spacing: 0.65em)
```

Reveal one after another: a list or several blocks.

Two notations, the same function:

```
#stagger[
  - this first
  - then this
]
#stagger(card[left], card[right])
```

For a list, the bullet marks are set here rather than left to `list`: only this way does the mark belong to the tracked element. If it stayed with the list, it would sit in the background and be there before its point appears.

`start` is `auto`: the sequence continues where the slide left off. `stride: 0` makes everything appear on the same step and staggers only through `stagger`, in milliseconds.

`dim: true` turns the sequence into a walk: the point being discussed stands there, the ones before it stay legible but muted. Every point then holds exactly its own step instead of the rest of the slide, and rests at `anim`'s `after: "dimmed"` from the next step on. Paging back brings each one up again.

Two things follow from that and are worth knowing before reaching for it. The last point dims too as soon as the slide has a further step after it, because then the walk has moved on from it as well. And `stride: 0`, which puts every point on one step, makes them all dim together on the next. A group that is revealed in whatever order it is called out.

For points that have no order of their own: what the class names gets shown, in the order it comes rather than the order it stands in. The digits `1` to `9` choose; the speaker view shows which digit belongs to which point.

```
#cue("ablesen", start: 2)[
  - positive und negative Werte
  - tiefster und höchster Wert
  - Abnahme und Zunahme
]
```

The group owns as many steps as it has points, and the order changes nothing about that. Everything that hangs on the step count – the progress bar, `info().step.total`, the overflow check, the handout – is therefore untouched.

Set, the list keeps its reading order: a point not yet named holds its place, so nothing jumps when it arrives later.

Name	Vorgabe
`name`	–
`..items`	–
`start`	`auto`
`spacing`	`0.65em`

15.3.6 `cue-layer`

```
cue-layer(name, number, body, enter: "fade")
```

Something that appears together with one point of an adaptive group.

A drawing layer, a picture, a sentence beside it: it shares the step with its point and therefore travels with it, without having to be linked.

```
#cue-layer("ablesen", 1, schicht-vorzeichen)
```

The group has to stand **before** its layers in the source, because a layer looks up which step its point was given. Standing after them, the package says so rather than quietly doing nothing.

Name	Vorgabe
`name`	–
`number`	–
`body`	–
`enter`	`"fade"`

15.3.7 `morph`

```
morph(
  name,
  body,
  at: "1-",
  duration: 900,
  match: "auto",
  inline: true,
)
```

Magic move: the same `name` twice, and the thing flies across.

Twice on two adjacent slides, or twice on one slide with `at:` selectors that do not overlap. The runtime pairs the flights step by step as well as slide by slide.

The name is a string or a label: `morph(<pythagoras>, …)`.

`duration` is 900 ms, not the duration of the presentation: a flight across the slide takes more time than a simple fade-in, and in real decks the default was overridden in 161 of 165 cases. `auto` falls back to the presentation's duration.

`match` is `"auto"`, `"glyph"` (always per glyph) or `"block"` (always as one rectangle).

Two names may be equal on the **target** slide. The runtime looks the source up by name but iterates over the targets, so two targets sharing a name both start from the same place and the glyph visibly splits in two. `at` is almost always right as it is. A morph is present from the first step on: a flight target must already be there when the slide is entered. Because paging back swaps the roles, that holds for both ends.

Delaying is only worthwhile for the **first** link in a chain, for example when the formula should appear together with its tile. It is allowed exactly when the preceding slide carries no morph of the same name; the package checks this at compile time and speaks up when it does not hold.

Under `prefers-reduced-motion: reduce` nothing flies. The slide changes the way it would change without a morph. See the manual.

Name	Vorgabe
`name`	–

Name	Vorgabe
`body`	–
`at`	`"1-"`
`duration`	`900`
`match`	`"auto"`
`inline`	`true`

15.3.8 `pin`

```
pin(name, body)
```

A named piece inside a morph.

Shape matching pairs glyphs by their outline and, where that is not enough, by proximity. Most of the time that is right. Where it is not, because the 3 in `3x^4` is meant to become the 3 in `4 dot 3x^3` and another 3 sits in between, the piece gets a name, and the pairing follows that instead.

```
#morph(<term>)[$#pin(<faktor>)[3] x^#pin(<hoch>)[4]$]
… and on the next slide …
#morph(<term>)[$#pin(<hoch>)[4] dot #pin(<faktor>)[3] x^3$]
```

Matching names find each other before the shape is consulted; everything else works as before. A pin with no counterpart on the other slide falls back to shape matching without complaint.

The name is a string or a label.

15.3.9 `scene`

```
scene(
  ..parts,
  stops: (),
  tween: 8,
  start: auto,
  width: 100%,
  height: 190pt,
  duration: auto,
  enter: "fade",
  still: auto,
  steady: auto,
)
```

A drawing as a function of a value, with stops for the talk.

```
#scene(
  x => tangent-at(f, x),
  stops: (-3, 0, 1.5, 3),   // four stops, three steps
  tween: 8,                 // frames between two stops
)
```

This is manim's `ValueTracker` turned around. There a number changes while the film runs and the picture follows it; here Typst draws at compile time and a number can only change at a step. So the deck writes a function from a value to a picture and says at which values the talk stops. Typst renders every stop and the frames in between, and a keypress pulls the picture from one stop to the next.

`stops` are the values themselves, not 0.0 to 1.0 – that is the whole difference to `flipbook`. The scene takes `stops.len() - 1` steps: the first stop is there as soon as the scene appears, every further one costs a keypress.

A stop may be a tuple, and then the drawing function takes that many arguments: `(a, b) => …` with `stops: ((1, 1), (1, 3), (2, 3))`. What is lost against manim is that there several trackers may move independently; here everything travels from stop to stop together.

`tween` is the number of frames **between** two stops. With `tween: 0` the scene jumps from stop to stop and shows nothing in between.

`duration` is the time one pull from stop to stop takes, not the time of the entrance – the same separation `morph` draws, and for the same reason: one is a journey, the other a fade.

The scene stands in a box of a fixed size and every frame is clipped to it. The scene stands in a box of a fixed size and every frame is clipped to it. Unlike `build` the frames are **not** laid out on top of one another: they are drawings of different values and may legitimately come out different sizes, so one shared frame is the only arrangement in which the box itself does not jump.

The frames are measured all the same, and `steady` says what that measurement is for. A CeTZ canvas is as large as what it holds, so a frame wider than its neighbour puts the drawing somewhere else inside the box, and paging through it the whole picture travels while only one point should move. The package can see that and cannot correct it: `measure` answers with a size, never with where the ink lies inside it.

- `auto`, the default: the frames are measured and a finding is filed as a record. `presentation(drift: …)` decides what happens with the records – `"error"`, the default, stops at the end of the deck with all of them at once.
- `false`: the frames are meant to differ – a rectangle that grows, a number that counts up – and this scene is taken out of the check. It is not measured at all.
- `true`: this scene has to stand still, and it stops where it stands if it does not, whatever the deck says.

Measuring costs one more layout per frame, and a frame is a whole layout. Measured on a scene of 28 frames – four stops, eight frames between each pair, a CeTZ drawing of axes with ticks, a parabola of 61 points, a tangent, a dashed slope triangle and two labels: 434 ms without the measuring and 536 ms with it, so about 100 ms for the scene and 3.6 ms per frame. Only the browser branch pays it. On paper a scene is one still image, and a still image does not travel.

On paper the last stop is set, as with `alternatives`; `still` overrides that. The step cursor still runs there, so `info().step.total` names the same number in both outputs.

Under `prefers-reduced-motion: reduce` the frames in between fall away and the scene jumps from stop to stop. That is the package's rule everywhere else too: what stays is the destination, what goes is the travel.

Name	Vorgabe
`..parts`	–
`stops`	`()`
`tween`	`8`
`start`	`auto`
`width`	`100%`
`height`	`190pt`
`duration`	`auto`
`enter`	`"fade"`
`still`	`auto`
`steady`	`auto`

15.3.10 `scene-layer`

```
scene-layer(name, nr, body, enter: "fade")
```

Something that belongs to one particular stop of a scene.

A sentence beside it, a formula, a second drawing: it shares the step with its stop and therefore travels with it, without having to be linked.

```
#scene("derivative", x => tangent-at(f, x), stops: (-3, 0, 3))

#scene-layer("derivative", 2)[At the vertex the slope is zero.]
```

The scene has to stand **before** its layers in the source, because a layer looks up which step its stop was given. Standing after them, the package says so rather than quietly doing nothing.

A layer stays from its stop to the end of the slide, as `cue-layer` does: what was said at a stop goes on holding afterwards.

Name	Vorgabe
`name`	–
`nr`	–
`body`	–
`enter`	`"fade"`

15.3.11 `stagger`

```
stagger(
  ..items,
  start: auto,
  stride: 1,
  enter: "fade-up",
  duration: auto,
  easing: auto,
  stagger: 60,
  spacing: 0.65em,
  dim: false,
  morph: false,
  name: none,
)
```

Several things, one after another, one step apart.

```
#stagger[
  - The first point
  - The second
  - And the third
]
```

A bullet list is taken apart at its items; anything else is taken as it comes, one piece per argument. Where a list would be wrong – three cards side by side, say – hand the pieces over instead:

```
#stagger(card[One], card[Two], card[Three])
```

`start` is the step the first piece stands on, `auto` the next free one. `stride` is how many steps lie between two pieces: `2` leaves one out, and `0` puts them all on the same step, staggered only by `stagger`, which is the delay in milliseconds between one piece and the next. The two belong together – with `stride: 0` and `stagger: 60` a list arrives as a wave rather than as a sequence of keypresses.

`dim` leaves the pieces already shown standing, dimmed, instead of at full strength. `spacing` is the gap between them, `enter`, `duration` and `easing` are handed on to every piece unchanged.

`morph: true` lets each piece fly out of the one before it. Every piece stays where it is once it has arrived, so at a step change the piece set last is the source and the new one the target: the new line grows out of the line above while the line above stays put. That is a chain of transformations, line by line, on a single slide:

```
#stagger(morph: true, spacing: 14pt,
  $ x^2 + 6 x + 2 = 0 $,
  $ (x + 3)^2 - 7 = 0 $,
  $ x = -3 plus.minus sqrt(7) $,
)
```

A morph has no entrance, no easing curve and no dimmed rest, so `enter:`, `easing:` and `dim:` are refused rather than quietly dropped. `duration:` is read and is the time of the flight.

Name	Vorgabe
`..items`	–
`start`	`auto`
`stride`	`1`
`enter`	`"fade-up"`
`duration`	`auto`
`easing`	`auto`
`stagger`	`60`
`spacing`	`0.65em`
`dim`	`false`
`morph`	`false`
`name`	`none`

15.3.12 `stagger-layer`

```
stagger-layer(name, number, body, enter: "fade")
```

Something that belongs to one particular piece of a `stagger`.

A note in the margin, an annotation on a line of a calculation, a second drawing: it shares the step with its piece and therefore travels with it, without having to be counted.

```
#stagger(morph: "rewrite", ..lines)

#stagger-layer("rewrite", 2)[$| -2$]
```

The group needs a name for that – `name:` says it, and a `morph:` written as a name says it too, because then it is already there.

The stagger has to stand **before** its layers in the source, because a layer looks up which step its piece was given. Standing after them, the package says so rather than quietly doing nothing.

A layer stays from its piece to the end of the slide, as `cue-layer` and `scene-layer` do. And it stays out of a `morph: true` flight: the layer is its own element and carries no morph name, so what flies is the piece and the annotation merely appears beside it – which is what an annotation should do.

Name	Vorgabe
`name`	–
`number`	–

Name	Vorgabe
`body`	–
`enter`	`"fade"`

15.3.13 `pause`

Everything after this appears a step later.

The short form for slides that simply unfold: no `anim` around anything, no step numbers.

```
== A slide
First this.
#pause
Then that.
```

It is read at the top level of the slide body, `#set` and `#show` rules included. Inside a grid cell, a table or a figure it is not seen: there the content is a field of an element, not part of the body. Reach for `anim` in those places instead.

15.4 Layouts

15.4.1 `callout`

```
callout(
  body,
  title: auto,
  color: auto,
  radius: 7pt,
  inset: (x: 14pt, y: 11pt),
  width: 100%,
)
```

A highlighted key sentence: Beamer's `alertblock`.

The bar on the left marks it on every slide at a glance as „the thing to remember“, without it looking like a second box.

Labelled `<ts-callout>`, `<ts-callout-title>` and `<ts-callout-body>`. As in `card`, the surface goes through a `set` rule, so a label rule reaches `fill`, `stroke` and `radius`.

Name	Vorgabe
`body`	–
`title`	`auto`
`color`	`auto`
`radius`	`7pt`
`inset`	`(x: 14pt, y: 11pt)`
`width`	`100%`

15.4.2 `card`

```
card(
  body,
  title: none,
  number: none,
  color: auto,
  fill: auto,
  stroke: auto,
  radius: auto,
  inset: (x: 12pt, y: 10pt),
  width: 100%,
)
```

A named box: Beamer's `block`.

`title` sits in a colored bar above it, `number` additionally places a numbered disc in front. Without either, a plain box remains.

`color`, `fill` and `stroke` take the theme's values on `auto`: its primary color, its card background and its border.

No `clip: true`: Typst derives a clip path's identifier from the content, and the same box twice on a slide would produce the same identifier. The corners are therefore rounded by the bar itself.

Six labels for the six parts, so a deck can restyle them without touching the theme. `<ts-card>` is the box itself, and because its surface travels through a `set` rule rather than an argument, `set block(fill: ..)` on that label reaches it. `<ts-card-bar>` is the coloured tab, `<ts-card-title>` the caption, `<ts-card-disc>` the numbered disc, `<ts-card-number>` the numeral in it, `<ts-card-body>` the body.

Name	Vorgabe
`body`	–
`title`	`none`
`number`	`none`
`color`	`auto`
`fill`	`auto`
`stroke`	`auto`
`radius`	`auto`
`inset`	`(x: 12pt, y: 10pt)`
`width`	`100%`

15.4.3 `fit`

```
fit(
  body,
  width: auto,
  height: auto,
  wrap: true,
  grow: false,
  shrink: true,
)
```

Scale one block down to the room it has.

For the thing whose size the deck does not control: a wide table, a generated diagram, a list that came out of a data file. Without it such a block runs over the edge of the slide. In the PDF it is still to be seen standing there; in the browser the slide sits in a frame of fixed size and whatever reaches past it is cut away.

```
#slide[
  == Regression results
  #fit(wrap: false, my-table)
]
```

`wrap: false` because the block is a table. Everything that lays itself out in columns has to be measured as it stands; see below.

The block is measured against the place it stands in and scaled geometrically, so it keeps its proportions and what stands around it counts with its new size. No factor is given by hand.

Width first, then smaller. The block is offered the full width before it is measured, so a paragraph or a list breaks into the space instead of shrinking, and only what is still too tall afterwards is scaled. A table, a chart or a drawing would rearrange its own columns instead, which changes the picture rather than its size; `wrap: false` measures such a block exactly as it stands.

It only shrinks. `grow: true` also blows a block up that is smaller than its place, for the one large number that is meant to fill the slide. `shrink: false` takes the shrinking away and leaves only the growing.

`width` and `height` take `auto`, a length or a ratio. `auto` is the whole place. On `height: auto` the block takes what is left over below whatever stands above it on the slide, so a fit under two bullet points reckons with the bullet points. That has a flip side wherever something encloses the fit: inside a `card` the box becomes slide-tall, is cut off at the bottom, and **whatever follows the card falls off the slide** – measured in both outputs. It is the `1fr` doing that, not the scaling: a `card` around a bare `block(height: 1fr)` behaves the same. Give `height:` explicitly inside a card, and the fit reckons with that instead.

No reveal inside. Two things do not survive being measured. A `pause` is found by walking the slide body, and a fitted block is a closure that walk cannot enter: measured on a slide carrying two pauses, the step count fell from three to one and nothing said so. And a measured block has no height to reckon against, which is the axis on which a tracked element resolves its size and reserves the room for its marker: measured, an `anim` inside a fit was not scaled at all and ran off the bottom of the slide. `fit` stops with a message instead, for `pause`, `anim`, `stagger`, `alternatives`, `morph`, `tiles`, `video`, `embed`, `flipbook`, `build`, `scene`, `camera` and `cue`, in both outputs. Fit what stands inside the reveal rather than fitting around it: `anim(fit(my-table))` works, `fit(anim(my-table))` is the error.

`speaker-note` and `bridge-job` are allowed inside a fit. They settle no geometry, and a `measure` commits no state, so both were measured to arrive exactly once. The other direction is the one that does not work: a note made only of a `fit` carries no text, and `speaker-note` refuses it.

The geometry is adapted from mosaic, which adapted Touying 0.7.4, which credits it to Andreas Kröpelin (Polylux PR 91) and to ntjess.

Name	Vorgabe
`body`	–
`width`	`auto`
`height`	`auto`
`wrap`	`true`
`grow`	`false`
`shrink`	`true`

15.4.4 `side-by-side`

```
side-by-side(
  ..parts,
  split: (1.25fr, 1fr),
  gutter: 18pt,
  align: horizon,
  equal: false,
)
```

Two or more columns side by side.

The name comes from Touying and Polylux, which chose it independently of each other.

`split` takes the column widths; the default gives the first column a bit more, because that is usually where the illustration sits and the text on the right.

Name	Vorgabe
`..parts`	–
`split`	`(1.25fr, 1fr)`
`gutter`	`18pt`
`align`	`horizon`
`equal`	`false`

15.4.5 `statement`

```
statement(
  body,
  size: 1.6em,
  color: none,
  above: 0.6em,
  below: 0.6em,
)
```

A large statement in the middle: the formula that matters.

Explicitly demands the full width: a tracked element becomes as wide as its content, and a bare `align(center, …)` inside it would have no room to center in and would sit unchanged on the left.

Labelled `<ts-statement>`. `size` is measured in `em`, so a `set text(size: …)` from a label rule multiplies rather than replaces it.

Name	Vorgabe
`body`	–
`size`	`1.6em`
`color`	`none`
`above`	`0.6em`
`below`	`0.6em`

15.4.6 `tiles`

```
tiles(
  ..items,
  columns: auto,
  gutter: 14pt,
  row-gutter: auto,
  at: auto,
  stride: 1,
  stagger: 0,
  enter: "fade-up",
  duration: auto,
  easing: auto,
  align: top + left,
)
```

A tile grid that staggers itself.

Each tile appears one step after the previous one: that is the reason this function exists. By hand that means an `anim` per tile with an incremented number or delay.

`at` behaves as in `anim`: `auto` takes the next free step. `stride: 0` makes all tiles appear on the same step and staggers only through `stagger`, in milliseconds; a wave then runs through the grid.

`duration` and `easing` are those of `anim` and apply to every tile alike: one grid moves as one thing. `auto` is the presentation's duration and the package's own curve. Without them a deck that wanted a slower tile or a curve with a swing had to leave the grid and write the `anim`s out by hand, which is the very work `tiles` exists to save.

Name	Vorgabe
`..items`	–
`columns`	`auto`
`gutter`	`14pt`
`row-gutter`	`auto`
`at`	`auto`
`stride`	`1`
`stagger`	`0`
`enter`	`"fade-up"`
`duration`	`auto`
`easing`	`auto`
`align`	`top + left`

15.5 Themes

15.5.1 `theme`

```
theme(
  paper: rgb("#fafafa"),
  ink: black,
  strong: rgb("#23303f"),
  accent: rgb("#eb5e28"),
  muted: luma(45%),
  surface: white,
  border: luma(84%),
  inverted: false,
  font: none,
  title-font: none,
  size: 24pt,
  title-size: 31pt,
  weight: "bold",
  tracking: 0pt,
  header: "band",
  title-fill: p => lesbar(p.strong, white, p.paper, p.ink),
  rule-size: 0pt,
  rule-fill: none,
  head-gap: 20pt,
  foot-gap: 24pt,
  band-height: 66pt,
  footer: "fraction",
  footer-rule: 0pt,
  progress: "bar",
  box: "bar",
  title-slide: band-title-slide,
  section: band-section,
)
```

Builds a theme. Without an argument it produces the default appearance: `themes.default` is exactly that.

The entries in groups: colors (`paper ink strong accent muted surface border inverted`), typography (`font title-font size title-size weight tracking`), the ordinary slide (`header title-fill rule-size rule-fill head-gap foot-gap band-height box footer footer-rule progress`) and the two whole pictures (`title-slide`, `section`).

The eight colors are also the entries a **palette** may carry, and that is how a theme is recolored without touching the rest of it. Two of the entries above are colors that are not palette entries, `title-fill` and `rule-fill`, and each may be written either as a color or as a function of the palette, `p => p.strong`. A color stays what it is under every palette; a function is asked again whenever one is applied, and that is what lets the title of a light theme follow into the dark. `rule-fill: none` means the accent and follows it.

Font sizes: measured against Beamer and Metropolis, where the body text takes up around 3.0% of the slide width and the title 3.9%. The earlier 19pt/23pt were at 2.3% and 2.7%: noticeably smaller than what you would want to read from the back row.

What the theme draws also carries labels, and those are the second way to reach it. Names follow one scheme, place first and part second. On an ordinary slide `ts-slide-ground`, `ts-slide-header-band`, `ts-slide-header-text`, `ts-slide-header-rule`, `ts-slide-title`, `ts-slide-title-rule`, `ts-slide-footer`,

`ts-slide-number`, `ts-slide-footer-rule`, `ts-slide-progress` and `ts-slide-progress-track`; on the title slide `ts-title-slide-ground`, `ts-title-slide-band`, `ts-title-slide-title`, `ts-title-slide-subtitle`, `ts-title-slide-rule` and `ts-title-slide-byline`; on the section slide `ts-section-slide-ground`, `ts-section-slide-bar`, `ts-section-slide-title`, `ts-section-slide-rule` and `ts-section-slide-parent`. A `show` rule on one of them changes type or fill without a key having to exist for it; the keys below stay what they are and keep the arrangement.

The last of those is the line naming the sections a deeper section hangs under. It exists only from the second structure level on, so a deck at the default `slide-level: 2` never draws it.

A theme that brings its own `title-slide` or `section` function draws none of those labels, and nothing warns about it. Such a `section` function receives the section record, and there `s.depth` is the heading level and `s.parents` are the titles above it, outermost first. A function that reads neither draws every level alike; nothing breaks, the hierarchy is simply not shown.

Comments must NOT go into the parameter list: tidy splits it at the commas and expects a colon in every piece; the API reference breaks on that.

Name	Vorgabe
`paper`	`rgb("#fafafa")`
`ink`	`black`
`strong`	`rgb("#23303f")`
`accent`	`rgb("#eb5e28")`
`muted`	`luma(45%)`
`surface`	`white`
`border`	`luma(84%)`
`inverted`	`false`
`font`	`none`
`title-font`	`none`
`size`	`24pt`
`title-size`	`31pt`
`weight`	`"bold"`
`tracking`	`0pt`
`header`	`"band"`
`title-fill`	`p => lesbar(p.strong, white, p.paper, p.ink)`
`rule-size`	`0pt`
`rule-fill`	`none`
`head-gap`	`20pt`
`foot-gap`	`24pt`
`band-height`	`66pt`
`footer`	`"fraction"`
`footer-rule`	`0pt`
`progress`	`"bar"`
`box`	`"bar"`
`title-slide`	`band-title-slide`
`section`	`band-section`

15.5.2 `themes`

The bundled themes: `themes.default`, `themes.lesson`, `themes.night`, `themes.plain`, `themes.editorial`.

They are made for different occasions, not the same slide in five colors: the title sits sometimes in a bar, sometimes free, sometimes under a line; the progress indicator grows, or is missing entirely.

15.6 Paletten

15.6.1 `contrast`

```
contrast(a, b)
```

The WCAG contrast ratio of two colours, a number from 1 to 21.

```
#contrast(black, white)                // 21
#contrast(rgb("#767b84"), white)       // 4.2539
```

The reference numbers WCAG 2 names: 4.5 for body text, 3.0 for large text and for lines, bars and other shapes that are not text. The package uses them in that sense in `palette-report`.

Alpha is ignored. A translucent colour is measured as if it were opaque, which is not what it looks like on the slide.

15.6.2 `palette-report`

```
palette-report(p)
```

Measure a palette against the contrast contract and report every pair.

```
#for f in palette-report(palettes.dark) [
  #f.pair: #calc.round(f.ratio, digits: 2) (wants #f.min) #f.ok \
]
```

One dictionary per pair, with `pair`, `ratio`, `min`, `ok` and `role`. It only measures and never changes a colour.

This is a report, not a gate. Only the five bundled palettes are actually held to the contract, by an assertion in this file; a palette written in a deck faces no such check, and none of the five bundled **themes** does either. Four of the five do not pass it, and that is deliberate rather than overlooked. See the manual.

15.6.3 `palettes`

The bundled palettes.

Five, one per colour world the package already had, and each of them composes with every theme: `themes.lesson` in `palettes.dark` is still the lesson design, only dark.

- `light` is exactly the default theme's colours, so `themes.default` with it is a no-op.
- `mono` is `themes.plain`'s greyscale, with two greys moved so it passes the contract.
- `textbook` is `themes.lesson`'s measured textbook colours, with `muted` moved for the same reason.
- `parchment` is `themes.editorial`'s laid paper, with `accent` and `muted` moved for the same reason.
- `dark` is `themes.night`'s dark ground with a deeper accent, because night's own cyan does not survive an inverted slide.

The six numbers that were moved, and why, are in the comments below.

15.7 Medien und Einbettungen

15.7.1 `embed`

```
embed(
  url: none,
  html: none,
  width: 100%,
  height: 200pt,
  at: "1-",
  enter: "fade",
  bridge: none,
  zoom: true,
  style: true,
  fallback: none,
  link: none,
  label: auto,
)
```

Arbitrary web content in a sandboxed frame.

`bridge` names the element so step jobs can be sent to it: that is how `geogebra` drives its applet, and how a companion package of your own would drive anything else, without the core knowing what is inside.

`fallback` and `link` only take effect in paged output; in the browser the embedded document itself stands there.

`style` gives a document passed as `html` the deck's basic style: it fills the frame, is transparent, and carries the running text size. Switched off, the frame is a blank browser page again.

Name	Vorgabe
`url`	`none`
`html`	`none`
`width`	`100%`
`height`	`200pt`
`at`	`"1-"`
`enter`	`"fade"`
`bridge`	`none`
`zoom`	`true`
`style`	`true`
`fallback`	`none`
`link`	`none`
`label`	`auto`

15.7.2 `flipbook`

```
flipbook(
  render,
  frames: 24,
  fps: 30,
  width: 200pt,
  height: 150pt,
  loop: true,
  pingpong: false,
  at: "1-",
  enter: "fade",
  still: auto,
)
```

Animation drawn by Typst, frame by frame.

`render` receives `t` running from 0.0 to 1.0. Every frame is rendered by Typst: CeTZ, Fletcher, equations, anything Typst can do. The frames sit in the file as SVG and stay sharp at any size.

Under `prefers-reduced-motion: reduce` it does not play. It stands on its last frame without `loop` and without `pingpong`, and on frame zero otherwise. See the manual.

Name	Vorgabe
`render`	–
`frames`	`24`
`fps`	`30`
`width`	`200pt`
`height`	`150pt`
`loop`	`true`
`pingpong`	`false`
`at`	`"1-"`
`enter`	`"fade"`
`still`	`auto`

15.7.3 `video`

```
video(
  src,
  width: 100%,
  height: 200pt,
  poster: none,
  autoplay: true,
  loop: false,
  muted: true,
  controls: false,
  radius: 0pt,
  at: "1-",
  enter: "fade",
)
```

A real HTML5 video over the slide.

Without a `poster:` the placeholder on paper is labelled `<ts-media-poster>`.

Name	Vorgabe
`src`	–

Name	Vorgabe
`width`	`100%`
`height`	`200pt`
`poster`	`none`
`autoplay`	`true`
`loop`	`false`
`muted`	`true`
`controls`	`false`
`radius`	`0pt`
`at`	`"1-"`
`enter`	`"fade"`

15.8 Die Brücke

15.8.1 `bridge-job`

```
bridge-job(target, payload, at: "1-")
```

Send a job to a bridged element.

- `target` is the `bridge` name given to `embed`.
- `at` is a step selector, resolved exactly as for `anim`: `auto` takes the next free step, an integer counts as „from this step on“. The default is `"1-"`, because most jobs set the document up on slide entry.

  A job does not move the step cursor: an applet's tween and the bullet explaining it usually belong on the **same** step, not one after the other.
- `payload` is a dictionary. Its meaning is entirely up to the document on the other side. The core passes it through unread.

When paging backwards or entering a slide the runtime replays the whole run from its start with a `reset` flag, so jobs should be repeatable.

The document has to announce itself. Nothing is sent to a frame that has not said hello. The runtime marks a frame live only after receiving

```
parent.postMessage({ typstage: 1, ready: 1 }, "*");
```

from it. Both fields are needed: the runtime drops every message without `typstage: 1` before it ever looks at `ready`. Miss this and the jobs simply never arrive. There is no error, the applet just sits there.

Name	Vorgabe
`target`	–
`payload`	–
`at`	`"1-"`

15.8.2 `bridge-targets`

```
bridge-targets()
```

The bridge names on the current slide, in the order they appear.

This is how a companion package can leave the applet unnamed: with exactly one bridged element on the slide there is nothing to choose between.

Must be called in a context. Duplicates are dropped, because a tracked element is laid out twice, once in the background and once as its sprite, so it announces itself twice.

15.9 GeoGebra

15.9.1 `geogebra`

```
geogebra(
  ..name,
  id: "ggb",
  material: none,
  app: "classic",
  perspective: "G",
  language: none,
  grid: auto,
  axes: auto,
  seamless: true,
  background: auto,
  animation-button: false,
  ,
  pan: false,
  font-size: 17,
  codebase: "https://www.geogebra.org/apps/",
  width: 100%,
  height: 330pt,
  at: "1-",
  fallback: none,
  link: none,
)
```

A GeoGebra applet in the slide.

- `seamless` takes the frame off the applet and puts its drawing area in the slide's colour, so it no longer looks like a window of its own.
- `fallback` and `link` only take effect in the PDF: there is no applet on paper, so what stands there is either a drawing of your own or at least the way to the live one.

The `id` is only needed when a slide holds more than one applet — the commands then say which one they mean. A single applet needs no name.

A `.ggb` file cannot be embedded: Typst has no base64 encoding, and without it the file content never reaches the HTML. Build the construction with `ggb-run` or load it through `material`.

Name	Vorgabe
`..name`	–
`id`	`"ggb"`
`material`	`none`
`app`	`"classic"`
`perspective`	`"G"`
`language`	`none`
`grid`	`auto`
`axes`	`auto`
`seamless`	`true`
`background`	`auto`
`animation-button`	`false`
	–

Name	Vorgabe
`pan`	`false`
`font-size`	`17`
`codebase`	`"https://www.geogebra.org/apps/"`
`width`	`100%`
`height`	`330pt`
`at`	`"1-"`
`fallback`	`none`
`link`	`none`

15.9.2 `ggb-animate`

```
ggb-animate(
  ..objects,
  target: auto,
  at: "1-",
  speed: none,
  playing: true,
  trace: (),
)
```

GeoGebra's own animation — it runs back and forth without end.

Name	Vorgabe
`..objects`	–
`target`	`auto`
`at`	`"1-"`
`speed`	`none`
`playing`	`true`
`trace`	`()`

15.9.3 `ggb-hide`

```
ggb-hide(..objects, target: auto, at: "1-")
```

Hide objects — the counterpart to `ggb-show`.

Name	Vorgabe
`..objects`	–
`target`	`auto`
`at`	`"1-"`

15.9.4 `ggb-run`

```
ggb-run(..commands, target: auto, at: "1-")
```

Run GeoGebra commands.

GeoGebra's scripting commands — `SetColor`, `SetValue` and relatives — are **not** accepted by `evalCommand` and would come to nothing here. That is what `ggb-style`, `ggb-set`, `ggb-show` and `ggb-hide` are for: they reach for the JavaScript interface, which can do it.

Name	Vorgabe
`..commands`	–
`target`	`auto`
`at`	`"1-"`

15.9.5 `ggb-set`

```
ggb-set(values, target: auto, at: "1-")
```

Set values in the applet.

Name	Vorgabe
`values`	–
`target`	`auto`
`at`	`"1-"`

15.9.6 `ggb-show`

```
ggb-show(..objects, target: auto, at: "1-")
```

Reveal objects that were hidden before.

Name	Vorgabe
`..objects`	–
`target`	`auto`
`at`	`"1-"`

15.9.7 `ggb-style`

```
ggb-style(
  ..objects,
  target: auto,
  at: "1-",
  color: none,
  thickness: none,
  line-style: none,
  filling: none,
  point-size: none,
  trace: none,
  label: none,
  label-mode: none,
  fixed: none,
  caption: none,
  layer: none,
  position: none,
)
```

Appearance — `color` takes a Typst colour, so the construction carries the colours of your slides instead of GeoGebra's palette.

Name	Vorgabe
`..objects`	–
`target`	`auto`
`at`	`"1-"`

Name	Vorgabe
`color`	`none`
`thickness`	`none`
`line-style`	`none`
`filling`	`none`
`point-size`	`none`
`trace`	`none`
`label`	`none`
`label-mode`	`none`
`fixed`	`none`
`caption`	`none`
`layer`	`none`
`position`	`none`

15.9.8 `ggb-tween`

```
ggb-tween(
  ..name,
  target: auto,
  to: 1.0,
  from: none,
  at: 1,
  duration: 650,
  easing: "ease-in-out",
)
```

Once from A to B — the construction draws itself.

GeoGebra's own animation runs back and forth for ever. Drawing needs the opposite — once from 0 to 1 and then stop — so here the browser counts the value up frame by frame. Build an object that depends on it and it grows along: a segment whose endpoint travels, an arc whose angle follows.

`at` points at **one** step — that is where it is drawn. On the steps after it the value simply sits at its target, so jumping back shows the finished drawing instead of the movement a second time.

Name	Vorgabe
`..name`	–
`target`	`auto`
`to`	`1.0`
`from`	`none`
`at`	`1`
`duration`	`650`
`easing`	`"ease-in-out"`

15.9.9 `ggb-view`

```
ggb-view(
  target: auto,
  at: "1-",
  x: none,
  y: none,
  grid: none,
  axes: none,
)
```

Viewport, grid, axes.

Name	Vorgabe
`target`	`auto`
`at`	`"1-"`
`x`	`none`
`y`	`none`
`grid`	`none`
`axes`	`none`

`desmos`

```
desmos(
  ..name,
  id: "dsm",
  api-key: none,
  version: "1.11",
  expressions: (:),
  bounds: none,
  ,
  expression-list: false,
  settings-menu: false,
  zoom-buttons: false,
  keypad: false,
  pan: false,
  grid: auto,
  axes: auto,
  axis-numbers: auto,
  seamless: true,
  background: auto,
  width: 100%,
  height: 330pt,
  at: "1-",
  fallback: none,
  link: none,
)
```

Ein Desmos-Graph auf der Folie.

- `api-key` ist Pflicht. Desmos gibt sein Skript ohne Schlüssel nicht heraus – gemessen antwortet der Server dann mit 403. Zum Ausprobieren gibt es `demo-key`; wer damit vorträgt, trägt Desmos' Konsolenwarnung mit sich herum und arbeitet außerhalb dessen, wofür der Schlüssel gedacht ist.
- `expressions` ist das Startbild als Wörterbuch von `id` nach `LaTeX`. Ein Ausdruck mit `=` ist ein Regler, einer ohne eine Kurve; das entscheidet Desmos.
- `bounds` ist der Ausschnitt als `(links, rechts, unten, oben)`.

Name	Vorgabe
`..name`	–
`id`	`"dsm"`
`api-key`	`none`
`version`	`"1.11"`
`expressions`	`(:)`
`bounds`	`none`
	–
`expression-list`	`false`
`settings-menu`	`false`
`zoom-buttons`	`false`
`keypad`	`false`
`pan`	`false`
`grid`	`auto`
`axes`	`auto`
`axis-numbers`	`auto`
`seamless`	`true`
`background`	`auto`
`width`	`100%`
`height`	`330pt`
`at`	`"1-"`
`fallback`	`none`
`link`	`none`

`dsm-animate`

```
dsm-animate(
  ..ids,
  target: auto,
  at: "1-",
  playing: true,
  min: none,
  max: none,
  step: none,
)
```

Desmos' eigene Regleranimation an- oder abschalten.

Sie läuft mit Desmos' Geschwindigkeit und ohne Ziel. Wer von einer Zahl zu einer anderen will und dann stehenbleiben, nimmt `dsm-tween`.

Name	Vorgabe
`..ids`	–
`target`	`auto`
`at`	`"1-"`
`playing`	`true`
`min`	`none`
`max`	`none`
`step`	`none`

`dsm-expr`

```
dsm-expr(..objects, target: auto, at: "1-")
```

Ganze Ausdrucksobjekte durchreichen, für alles, was `dsm-set` nicht kann – Farbe, Reglergrenzen und Stil in einem Zug.

Name	Vorgabe
`..objects`	–
`target`	`auto`
`at`	`"1-"`

`dsm-hide`

```
dsm-hide(..ids, target: auto, at: "1-")
```

Ausdrücke verbergen. Sie bleiben im Graphen und rechnen weiter mit.

Name	Vorgabe
`..ids`	–
`target`	`auto`
`at`	`"1-"`

`dsm-remove`

```
dsm-remove(..ids, target: auto, at: "1-")
```

Ausdrücke entfernen.

Name	Vorgabe
`..ids`	–
`target`	`auto`
`at`	`"1-"`

`dsm-set`

```
dsm-set(values, target: auto, at: "1-")
```

Ausdrücke setzen oder ändern – ein Wörterbuch von `id` nach `LaTeX`.

Dieselbe `id` noch einmal ersetzt den Ausdruck, statt einen zweiten anzulegen. So bewegt sich eine Kurve über die Schritte.

Name	Vorgabe
`values`	–
`target`	`auto`
`at`	`"1-"`

`dsm-show`

```
dsm-show(..ids, target: auto, at: "1-")
```

Ausdrücke zeigen.

Name	Vorgabe
`..ids`	–
`target`	`auto`
`at`	`"1-"`

`dsm-style`

```
dsm-style(
  ..ids,
  target: auto,
  at: "1-",
  color: none,
  line-style: none,
  line-width: none,
  line-opacity: none,
  point-style: none,
  point-size: none,
  fill: none,
  fill-opacity: none,
  label: none,
  show-label: none,
  drag-mode: none,
)
```

Aussehen eines Ausdrucks. Die Schlüssel sind Desmos' eigene.

Name	Vorgabe
`..ids`	–
`target`	`auto`
`at`	`"1-"`
`color`	`none`
`line-style`	`none`
`line-width`	`none`
`line-opacity`	`none`
`point-style`	`none`
`point-size`	`none`
`fill`	`none`
`fill-opacity`	`none`
`label`	`none`
`show-label`	`none`
`drag-mode`	`none`

`dsm-tween`

```
dsm-tween(
  name,
  target: auto,
  to: 1.0,
  from: none,
  at: 1,
  duration: 600,
  easing: "ease",
)
```

Einen Regler von einer Zahl zur anderen ziehen, in einer gegebenen Zeit.

Zwei Aufträge, wie bei `ggb-tween` nebenan, und der zweite ist der wichtigere: Die Bewegung liegt auf **genau** diesem Schritt, und ab dem nächsten steht der Endwert einfach da. Ohne das liefe sie auf jedem weiteren Schritt der Folie noch einmal an – gemessen sprang der Regler beim Weiterblättern von 3 zurück auf 0,75 und wuchs erneut, weil ein ganzzahliges `at` zu „ab diesem Schritt“ wird und nicht zu „auf diesem“.

Auf Papier und beim Zurückblättern steht das Ergebnis sofort da: eine Bewegung, die niemand sieht, ist keine.

Name	Vorgabe
`name`	–
`target`	`auto`
`to`	`1.0`
`from`	`none`
`at`	`1`
`duration`	`600`
`easing`	`"ease"`

`dsm-view`

```
dsm-view(
  target: auto,
  at: "1-",
  bounds: none,
  grid: none,
  axes: none,
  axis-numbers: none,
  degrees: none,
  polar: none,
)
```

Der Ausschnitt, als `(links, rechts, unten, oben)`, und die Grapheinstellungen.

Name	Vorgabe
`target`	`auto`
`at`	`"1-"`
`bounds`	`none`
`grid`	`none`
`axes`	`none`
`axis-numbers`	`none`
`degrees`	`none`
`polar`	`none`

`demo-key`

Der Demo-Schlüssel aus Desmos’ eigener Dokumentation.

Er ist ausdrücklich zum Ausprobieren gedacht, nicht für den Betrieb. Das geladene Skript sagt es selbst in der Konsole: „This page is using the Desmos API with a trial key suitable for prototyping, not for commercial use.“ Einen eigenen gibt es über https://www.desmos.com/my-api.

15.10 Maße, Farben, Laufzeitdateien

15.10.1 `dark`

The default palette. Override it by wrapping the presentation in your own document template. See `style` on `presentation`.

15.10.2 `runtime-files`

The runtime files, ready to be written next to the HTML.

Typst cannot create files. Whoever uses `assets: "split"` or a CDN writes them out once. The content comes from here so the copies cannot drift.

15.10.3 `runtime-version`

Version of the runtime. It goes into the asset file names so a CDN can hold several releases side by side and no browser serves a stale one from cache.

15.10.4 `slide-width`

Default slide geometry. 16:9 on an A4-width canvas, so a slide and a handout page carry text at the same physical size. `presentation` takes `width`, `height` and `margin` to override them.