Auditorium turns a folder of plain HTML, CSS and JavaScript into a talk you present, a poster, a pitch deck, a printed report, a dashboard, a narrated explainer video or a vertical short. You write the page; Auditorium presents, previews, checks, renders and exports it.

Time lives in the HTML. A `<section>` is a beat (a slide, a scene), and `data-` attributes on its elements say when each one enters, how it moves and when it leaves:

```
<main class="aud" data-format="deck">
  <section data-beat="claim" class="statement">
    <p data-enter="words">Small steps beat <em>big guesses</em>.</p>
  </section>
  <section data-beat="steps">
    <h2>Three steps</h2>
    <ol>
      <li data-step="1" data-enter="rise">Measure the slope</li>
      <li data-step="2" data-enter="rise">Take a small step down</li>
      <li data-step="3" data-enter="rise">Repeat</li>
    </ol>
  </section>
</main>
```

When the page loads, the runtime reads those attributes once and builds a timeline. One function, `seek(t)`, puts the page in the state it holds at time `t`. Presenting, the authoring preview, the presenter view, video rendering and every export call that same function, so the video you render is the page you previewed, frame for frame.

The `auditorium` command is the tool around the folder:

COMMAND	WHAT IT DOES
`auditorium template`	lists, shows and copies the seven starting projects
`auditorium check`	checks a project against every rule, with the line and the fix
`auditorium preview`	the authoring view: scrubber, frame stepping, loops, and `--edit` to edit in place
`auditorium run`	presents live, with a presenter view and an optional public URL
`auditorium sheet`	one contact-sheet PNG of the whole timeline
`auditorium export`	PNG stills, a PDF, or one self-contained HTML file
`auditorium bundle`	one self-contained HTML file that presents from disk
`auditorium render`	an MP4, WebM or PNG sequence, frame by frame, deterministic
`auditorium assets`	resolves `assets.yaml`: files, page captures, generated images
`auditorium music`	composes a music bed
`auditorium relay`	runs your own relay for public presenting

COMMAND	WHAT IT DOES
`auditorium help`	the guide that ships with the package, one topic at a time

It is written for people and for agents. A small model with a shell and this CLI can go from an empty folder to a finished artifact: the templates mark what to replace, the check names the line and the fix, and `auditorium help` holds the reference for the installed version.

This book is version 5. Version 4 and earlier were a Python API (`Deck`, `@deck.scene`); 5.0 removed it, and nothing here uses it.

How to read this book

- **Quick Start** builds a two-slide deck from an empty folder and ships it as one HTML file.
- **The User Guide** has one chapter per thing people make: a talk, a poster, a pitch, a report page, a dashboard, an explainer video and a short. Each starts from its template and ends with the artifact.
- **The Gallery** links every finished example as a self-contained HTML file you can open, play and step through in your browser.
- **The Developer Guide** explains how the runtime, the server and the tools fit, and how to add a template, a check rule, a motion or an image provider.

Every command in this book ran when the book was built, against the version of Auditorium in the repository, and its output is printed below it. The book is also a PDF.

1

Quick Start

The job: a two-slide deck that says one thing, checked, previewed and shipped as a single HTML file you can send to anyone.

Install

Version 5 is not on PyPI yet (`pip install auditorium` still installs 4.x), so run it from a checkout. Auditorium drives Chromium through Playwright for the check, the stills and the video, so install a browser once:

```
git clone https://github.com/gia-uh/auditorium && cd auditorium
uv sync
uv run playwright install chromium
uv run auditorium --version
```

The rest of this book writes `auditorium` for `uv run auditorium`.

Write the page

A project is a folder with an `index.html`. The page loads the runtime from `/_auditorium/`, a path the Auditorium server and the bundler both provide, and says its format on `<main class="aud">`. A `deck` is a 16:9 stage that the presenter advances by hand.

examples/hello/index.html · L1–25

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Hello, Auditorium</title>
  <link rel="stylesheet" href="/_auditorium/auditorium.css">
  <link rel="stylesheet" href="theme.css">
  <script type="module" src="/_auditorium/auditorium.js"></script>
</head>
<body>
<main class="aud" data-format="deck">
  <section data-beat="claim" class="statement">
    <p data-enter="words">Small steps beat <em>big guesses</em>.</p>
  </section>
  <section data-beat="steps">
    <h2>Three steps</h2>
    <ol>
      <li data-step="1" data-enter="rise">Measure the slope</li>
      <li data-step="2" data-enter="rise">Take a small step down</li>
      <li data-step="3" data-enter="rise">Repeat until it stops moving</li>
    </ol>
  </section>
```

```
</main>
</body>
</html>
```

Each `<section>` is a beat. `data-enter="words"` brings the sentence in a word at a time. `data-step="1"` holds a list item back until the first press of the presenter, so the second slide takes three presses.

Colours come from a theme, a file of CSS variables. The page names roles such as the accent (the `<em>` above), and the theme picks the colour:

examples/hello/theme.css · L1–6

```
:root {
  --aud-slide-bg: #fbfaf7;
  --aud-fg: #1d1d1f;
  --aud-muted: #6b6b70;
  --aud-accent: #c2410c;
}
```

Check it

`auditorium check` reads the files, then opens the page in Chromium and seeks it to every stop, measuring what shows: overflow, overlap, contrast, type size, script errors. It exits 1 while any error remains, and each finding names its line and its fix.

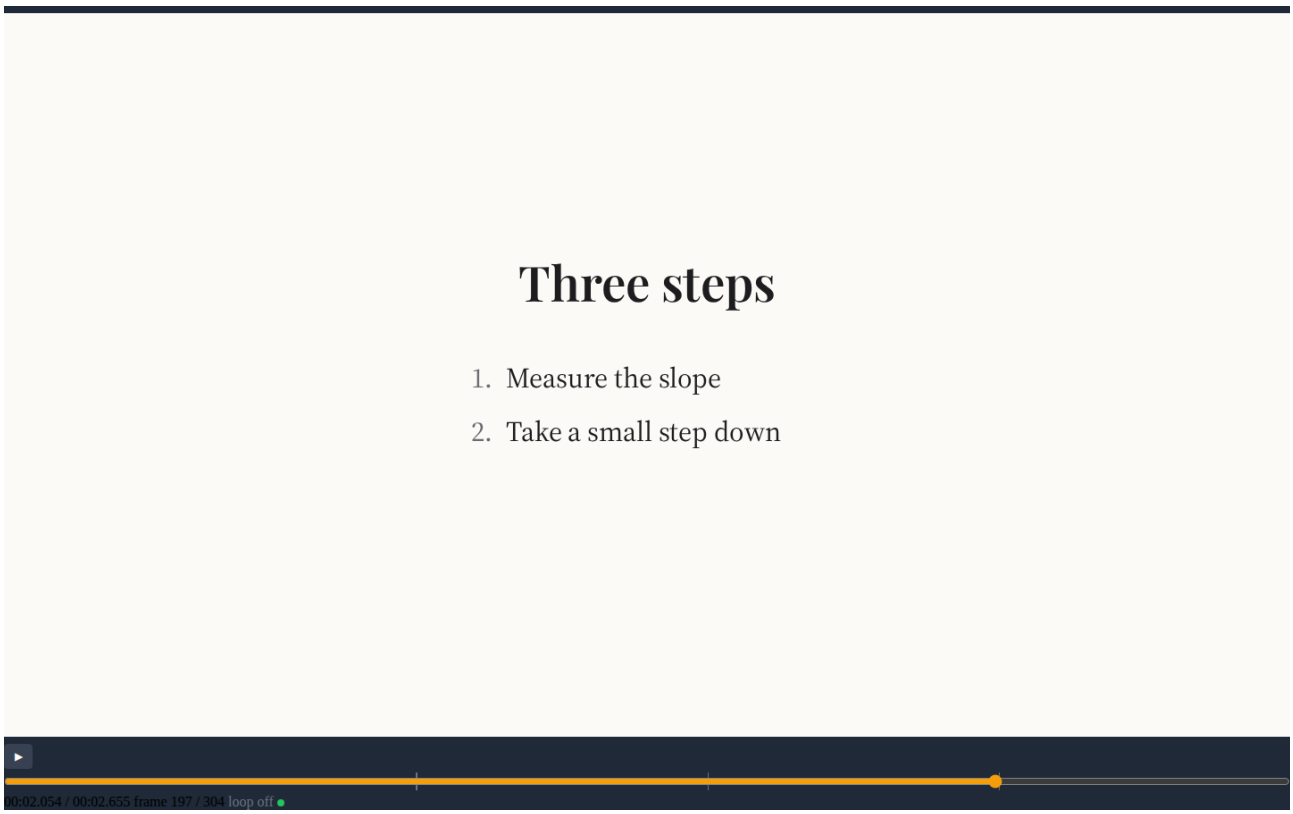
```
auditorium check examples/hello
```

```
0 errors, 0 warnings (structure and layout checked)
```

Look at it

`auditorium preview` serves the folder and opens the authoring view: the page on its stage, a scrubber with a tick at every stop, frame stepping with `.` and `,`, and a loop over a range with `i`, `o` and `l`. Saving a file reloads the page at the same instant.

```
auditorium preview examples/hello
```



Ship it

`auditorium bundle` writes the project as one HTML file with the runtime, the CSS, the fonts and every asset inside. It presents from disk with the arrow keys, space or a tap, and loads nothing from the network.

```
auditorium bundle examples/hello -o build/gallery/hello.html
```

```
✓ build/gallery/hello.html (font 0.16 MB, script 0.06 MB, style 0.25 MB)
333842 bytes in 1 file
```

Open the result and press the right arrow to step through it.

What to read next

Most work starts from a template rather than an empty folder, because a template already has the layouts, the logo slot and the beats of its kind of artifact. The next chapter starts a talk that way. For the reference of the version you have installed, ask the CLI:

```
auditorium help | tail -14
```

```
`auditorium help TOPIC` prints one topic; `auditorium help --all` prints them all.

assets      assets.yaml, the files, page captures and generated images a project uses
attributes  the file contract, every data- attribute and what it means
check       how to read the check's findings, and the rules it runs
components  the pieces inside a layout, as classes on plain HTML
export      one HTML file, a PDF, PNG stills, a contact sheet
formats     the seven formats, their stages and safe areas
layouts     ten beat layouts, each a class on a <section>
motion      every entrance, exit, idle, emphasis and layer preset, with its defaults
start       the loop, from an empty folder to an artifact
templates   which template for which request
themes      the theme variables, the bundled themes and the logo slot
11 topics; auditorium help TOPIC prints one
```

PART I

User Guide

2

A talk deck

The job: a forty-minute lecture on gradient descent, presented live from a laptop, with speaker notes on a second screen, a formula, some code, a figure that draws itself, and a PDF of the slides for the students afterwards.

Start from the template

A template is a finished kind of artifact with its content marked for replacement. Seven ship with Auditorium:

```
auditorium template list
```

```
dashboard  KPI tiles, a chart, a table and filter chips over data (page)
           A dashboard someone opens in a browser: three key numbers, a bar chart, a
table and chips that filter them, drawn from data embedded in the page or fetched live
from an endpoint declared in auditorium.toml.
explainer  a narrated landscape video, 1 to 6 minutes (explainer)
           A video that explains one idea out loud: a question, a statement, a figure, a
comparison, numbers, a closing line, each beat with its narration clip. Rendered to mp4.
pitch      an institutional or product deck (deck)
           A pitch to partners, clients or funders: big statements, stats, a photo, part-
ner logos, a closing ask. The presenter advances it.
poster     a single still for a feed or a print (square, portrait, story)
           An announcement, a result or an event: a kicker, a headline, a line of text,
up to three numbers. Exported as one PNG.
report     a long document with figures, printed to A4 (page)
           A report, a brief or a write-up someone reads on a screen or on paper: a
title, sections of prose, a figure, a table, a chart. Exported as an A4 PDF.
short      a vertical video, 15 to 60 s (short)
           A clip for Reels, TikTok or Shorts: a hook, a number, a line, a call to ac-
tion, over a music bed that ducks under optional narration. Rendered to mp4.
talk       a technical talk, presented live (deck)
           A lecture or conference talk: title, parts, statements, steps, a figure, math,
code. The presenter advances it.
7 templates; auditorium template show NAME prints one
```

`template show` prints what a template is for and the `index.html` it copies; `template new` copies it into a new folder. The folder is yours from then on: a later version of Auditorium never rewrites it.

```
auditorium template show talk | head -4
auditorium template new talk build/my-talk
```

```
talk: a technical talk, presented live
A lecture or conference talk: title, parts, statements, steps, a figure, math, code. The
presenter advances it.
formats: deck (the first is the default)
files: assets/logo.svg, index.html, theme.css, plus auditorium.toml and README.md
  build/my-talk/README.md
  build/my-talk/assets/logo.svg
  build/my-talk/auditorium.toml
  build/my-talk/index.html
  build/my-talk/theme.css
5 files written to build/my-talk from template talk, format deck; next: auditorium pre-
view build/my-talk
```

Replace what the template marks

Every element you must replace carries `data-placeholder`, and the check fails while one remains, so a half-edited deck cannot pass as finished:

```
auditorium check build/my-talk | tail -6
echo "exit code $?"
```

```
index.html:41  placeholder  error
  section[data-beat="steps"] h2
  "Three steps" is template text to replace
  fix: write the real content, then delete data-placeholder
... 11 more in dist/check.json
19 errors, 0 warnings (structure and layout checked)
exit code 0
```

The `exit code` line prints `0` here because `tail` ran last; the check itself exits `1`. Each finding names the line, the element, what is wrong and the fix. Write the content, delete the attribute, and run the check again until it prints `0 errors`.

The package keeps each template finished, under `example/`. The talk's example is the gradient-descent lecture, and the rest of this chapter works on a copy of it:

```
EX=$(python -c "import auditorium.templates as t; print(t.TEMPLATES)")
cp -r "$EX/talk/example" build/talk
auditorium check build/talk
```

```
0 errors, 0 warnings (structure and layout checked)
```

Steps, notes, math and code

A deck has no clock: the presenter advances it. Each press plays to the next stop, and a stop is either the start of a beat or a `data-step` inside it:

```
<section data-beat="title" class="title">
  <span class="kicker">Lecture 3, Machine Learning</span>
  <h1 data-enter="words">Gradient descent, one step at a time</h1>
  <p data-in="0.8" data-enter="fade">How a ball finds the bottom of a valley</p>
  <aside class="notes">Open with the picture of a ball rolling downhill.</aside>
</section>
<section data-beat="steps">
  <h2>The loop</h2>
  <ol>
    <li data-step="1" data-enter="rise">Measure the slope where you stand</li>
    <li data-step="2" data-enter="rise">Take a small step downhill</li>
    <li data-step="3" data-enter="rise">Repeat until the ground is flat</li>
  </ol>
</section>
```

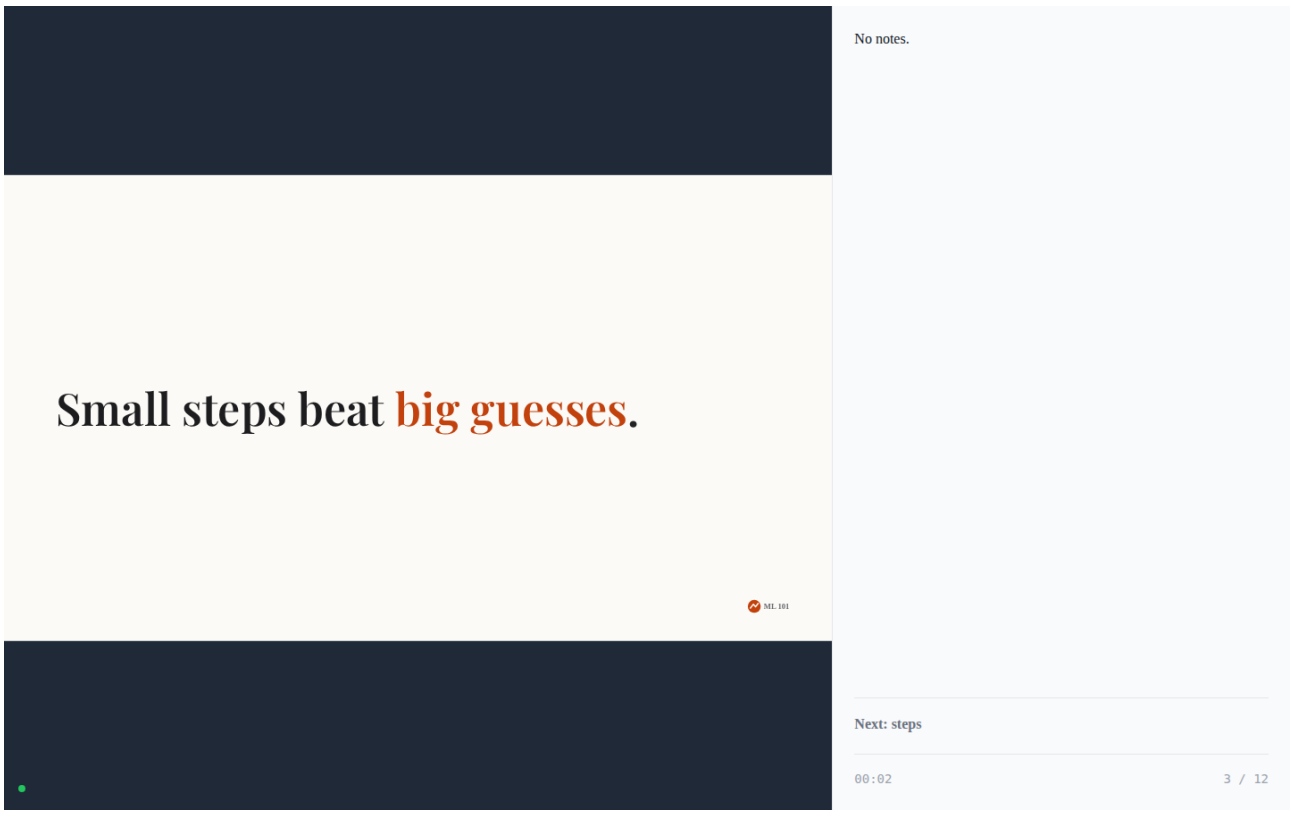
`data-in="0.8"` delays the subtitle 0.8 s after its beat starts. `<aside class="notes">` is only shown in the presenter view. Math is TeX inside `class="math"`, drawn by a copy of KaTeX that ships with the package, and code is a `<pre data-lang="python">`, highlighted by a vendored highlighter, so nothing is fetched while you present:

````
<div class="math" data-enter="fade">\theta_{t+1} = \theta_t - \eta \, \nabla
f(\theta_t)</div>
``` The beat layouts (`title`, `statement`, `figure`)
are classes on the `<section>`; `auditorium help layouts` prints all ten.

## Present it

`auditorium run` serves the deck and opens it. With `--presenter` it opens two
tabs, the audience view and the presenter view with the notes, a timer and the
next beat. Every connected browser follows the presenter.

```bash
auditorium run build/talk --presenter
````



The right arrow, space and Page Down play to the next stop; the left arrow goes back; `f` is full screen. A tap on the right of a touch screen goes forward.

To let people follow from their own devices, `--public` connects the deck to a relay and prints a URL:

```
auditorium run build/talk --public --name gradient-descent
```

The relay keeps no copy of the deck: it fetches each file a viewer asks for from your machine, so an edit you save reaches the room. You can run your own relay with `auditorium relay` and point the deck at it with `--relay HOST:PORT`. Speaker notes are part of `index.html`, so anyone with the URL can read them in the page source.

Hand it out

`auditorium sheet` puts every beat at its end, and frames between them, into one labelled PNG. It is the fastest way to see a whole deck after an edit:

```
auditorium sheet build/talk --beats | tail -4
```

```
frame    660     7.010 s  math
frame    720     7.511 s  code
frame    787     8.272 s  end
```

```
8 tiles in /home/runner/work/auditorium/auditorium/docs/build/talk/dist/sheet.png; audit-
orium render --from F --to F+1 reproduces frame F
```

`export -f pdf` prints one page per beat with every step shown, as text you can select and search:

```
auditorium export build/talk -f pdf -o build/talk.pdf
```

```
✓ build/talk.pdf
8 pages
```

And `export -f html` writes the deck as one file that presents from disk:

```
auditorium export build/talk -f html -o build/gallery/talk.html
```

```
✓ build/gallery/talk.html (font 0.42 MB, image 0.00 MB, script 0.06 MB, style 0.25 MB)
1104302 bytes in 1 file
```

Open the talk and step through it with the arrow keys.

3

A poster

The job: an announcement for an open lab day, as one image for a social feed, and the same poster cut for stories.

A still is a beat with no time

The poster template has one beat and no motion. Its format says the stage size, and the template offers three:

```
auditorium template show poster | head -3
```

```
poster: a single still for a feed or a print
An announcement, a result or an event: a kicker, a headline, a line of text, up to three
numbers. Exported as one PNG.
formats: square, portrait, story (the first is the default)
```

FORMAT	STAGE	FOR
`square`	1080 × 1080	a feed post
`portrait`	1080 × 1350	a tall feed post
`story`	1080 × 1920	stories and status

`--format` picks one when you copy the template. The layouts read the stage, so the same markup sets its type and margins for each size:

```
auditorium template new poster build/poster-story --format story | tail -1
grep -n 'data-format' build/poster-story/index.html
```

```
5 files written to build/poster-story from template poster, format story; next: auditori-
um preview build/poster-story
16:<main class="aud" data-format="story">
```

The finished poster

The example is the lab-day announcement. Its whole beat is a kicker, a headline, a line and two numbers:

```
<main class="aud" data-format="square">
  <img class="brand" src="assets/logo.svg" alt="AI lab">
  <section data-beat="poster" class="title">
    <span class="kicker">AI lab, Friday 14:00</span>
    <h1>Open lab day</h1>
    <p>See what the students built this term, and try it yourself</p>
    <div class="stats">
      <div class="stat"><b>12</b><span>projects</span></div>
      <div class="stat"><b>3</b><span>live demos</span></div>
    </div>
  </section>
</main>
```

`<img class="brand">` is the logo slot: it sits in the corner the layouts leave free, on every beat of every template.

Export the image

`export -f png` writes each beat at its end state. A poster has one beat, so it writes one file; `--size` renders the same stage at another resolution.

```
EX=$(python -c "import auditorium.templates as t; print(t.TEMPLATES)")
cp -r "$EX/poster/example" build/poster
auditorium check build/poster
auditorium export build/poster -f png -o build/poster-png
ls build/poster-png
```

```
0 errors, 0 warnings (structure and layout checked)
✓ 1 PNG files in build/poster-png/
beat-01.png
```



To cut it for stories, change `data-format="square"` to `story` in your copy and export again; the check tells you if a line no longer fits.

```
auditorium export build/poster -f html -o build/gallery/poster.html
```

```
✓ build/gallery/poster.html (font 0.16 MB, image 0.00 MB, script 0.06 MB, style 0.25 MB)
334578 bytes in 1 file
```

Open the poster.

4

A pitch

The job: a six-slide deck asking funders to put solar panels on fifty more schools, in the co-operative's colours, with its partners' logos, a photo, and numbers that count up as they appear.

Big statements and counting numbers

A pitch is a deck like a talk, built from fewer words and larger type. The template's layouts carry most of it: `statement` for one sentence, `stats` for up to three numbers, `photo`, and `logos` for a row of partners.

```
<section data-beat="numbers" class="stats">
  <h2>What we have built so far</h2>
  <div class="stat" data-enter="count"><b data-to="42">0</b><span>schools on solar</
span></div>
  <div class="stat" data-in="0.3" data-enter="count"><b data-to="1.8">0</b><span>MW in-
stalled</span></div>
  <div class="stat" data-in="0.6" data-enter="count"><b data-to="96">0</b><span>% fewer
lost hours</span></div>
</section>
```

`data-enter="count"` counts the number from the text it starts with (`0`) to `data-to`, and keeps the decimals `data-to` has. The `data-in` values stagger the three tiles by 0.3 s.

A theme for the brand

A theme is a file of CSS variables only: the palette, the fonts, the radius, the shadow and the code colours. Twelve ship with Auditorium, and `--theme` links one when you copy a template:

```
auditorium template new pitch build/pitch-dark --theme dark | tail -1
grep -n 'themes/' build/pitch-dark/index.html
```

```
6 files written to build/pitch-dark from template pitch, format deck; next: auditorium
preview build/pitch-dark
8:  <link rel="stylesheet" href="/_auditorium/themes/dark.css">
```

The project's own `theme.css` is linked after it, so any variable you set there wins. Each bundled theme passes WCAG AA contrast on every template's example; `auditorium help themes` lists the variables.

Bring the logos in

The partners' logos live somewhere else on your disk. Rather than copying them by hand, list them in `assets.yaml`, keyed by the path the page uses, and let `auditorium assets` copy them in. It keeps each file in a cache under its hash, so a second run fetches nothing, and it is the only command that reads outside the folder.

```
EX=$(python -c "import auditorium.templates as t; print(t.TEMPLATES)")
cp -r "$EX/pitch/example" build/pitch
cp build/pitch/assets/partner-1.svg build/brand/coastal.svg
cat > build/pitch/assets.yaml <<EOF
assets/partner-1.svg: {file: $PWD/build/brand/coastal.svg}
EOF
auditorium assets build/pitch
```

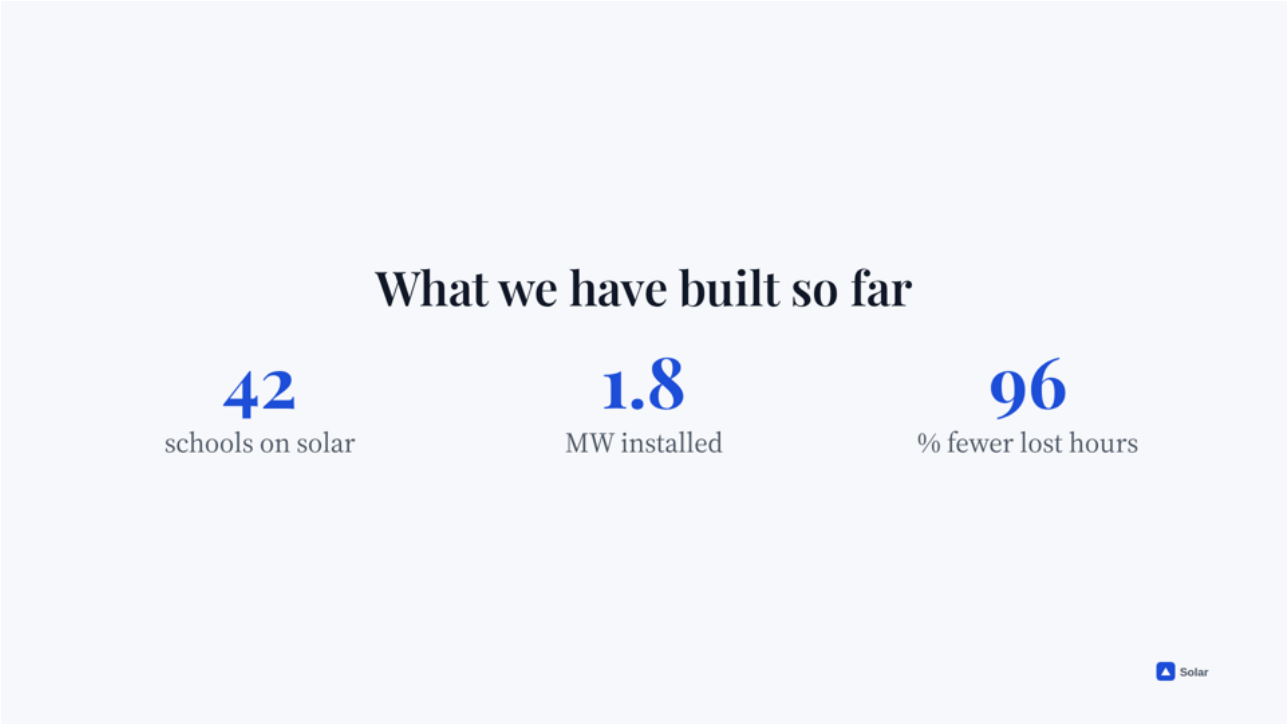
```
resolved     assets/partner-1.svg  (file)
1 resolved, 0 placeholders, 1 entries
```

An entry can also capture a web page as a screenshot (`capture:`) or ask an image provider plugin for a picture (`generate:`); `auditorium help assets` has both. An entry that cannot resolve becomes a labelled placeholder of the right size, and the check fails on it until it is fixed.

Check, look, send

```
auditorium check build/pitch
auditorium export build/pitch -f png -o build/pitch-png --beat numbers
```

```
0 errors, 0 warnings (structure and layout checked)
✓ 1 PNG files in build/pitch-png/
```



A pitch is often read on a phone after the meeting. `bundle` writes the one file to send; it refuses, writing nothing, if any file is missing or anything would load from the network, and names the line that does.

```
auditorium bundle build/pitch -o build/gallery/pitch.html
```

```
✓ build/gallery/pitch.html (font 0.16 MB, image 0.00 MB, script 0.06 MB, style 0.25 MB)
337432 bytes in 1 file
```

Open the pitch.

5

A report page

The job: a short report on water use in a district's schools, read on a phone by the school heads and printed at A4 for the council meeting, from one file.

A page has no stage

Decks and videos have a fixed stage that scales to the screen. A report should instead flow like a web page, so its format is `page`: every section shows at once, there is no clock and no presses, and the layouts stack their columns on a narrow screen. The check fails a page that scrolls sideways at 390 px, the width of a small phone.

```
auditorium template show report | head -3
```

```
report: a long document with figures, printed to A4
A report, a brief or a write-up someone reads on a screen or on paper: a title, sections
of prose, a figure, a table, a chart. Exported as an A4 PDF.
formats: page (the first is the default)
```

The sections are still `<section data-beat>`, so the same layouts work: a `figure` for an SVG chart drawn in the theme's colours, a `table`, and a `chart` whose bars are plain elements sized by a CSS variable:

```
<section data-beat="chart" class="chart">
  <h2>What each repair saves</h2>
  <div class="bars">
    <div class="bar" style="--v: 1"><span>cisterns</span><i></i><b>8,400 m³</b></div>
    <div class="bar" style="--v: 0.55"><span>taps</span><i></i><b>4,600 m³</b></div>
    <div class="bar" style="--v: 0.36"><span>meters</span><i></i><b>3,000 m³</b></div>
  </div>
</section>
```

`--v` is the bar's length as a fraction of the longest. The SVG figure uses `class="stroke-accent"` and `class="fill-ink"` instead of colours, so it follows the theme.

On a phone and on paper

```
EX=$(python -c "import auditorium.templates as t; print(t.TEMPLATES)")
cp -r "$EX/report/example" build/report
auditorium check build/report
```

```
0 errors, 0 warnings (structure and layout checked)
```

Water

DISTRICT WATER OFFICE, MARCH 2027

Water use in the district's schools

Where the water goes, and the three repairs that save the most

Summary

The district's 24 schools used 61,000 cubic metres of water in 2026, a fifth more than in 2024. Most of the rise comes from leaks in old buildings, not from more pupils: the six schools built before 1970 use twice as much water per pupil as the rest. Fixing leaking cisterns, replacing the oldest taps and metering each building

`export -f pdf` prints a page format through its print styles at A4. The text stays text, so the PDF can be searched and quoted:

```
auditorium export build/report -f pdf -o build/report.pdf
```

```
✓ build/report.pdf
3 pages
```

The same folder bundles into the file you send to the school heads:

```
auditorium bundle build/report -o build/gallery/report.html
```

```
✓ build/gallery/report.html (font 0.16 MB, image 0.00 MB, script 0.06 MB, style 0.25 MB)
337298 bytes in 1 file
```

Open the report, then make the window narrow.

6

A dashboard

The job: a page a cooperative's members open to see the year's sales by region, with chips that filter by quarter. It must open offline with the data it shipped with, and update when the sales service answers.

Drawing from data

A dashboard is a `page` like a report, plus a script. The rows ship inside the page as JSON, so it draws with no network at all:

```
<script type="application/json" id="data">
[
  {"group": "Q1", "label": "North", "value": 41000, "count": 520},
  {"group": "Q1", "label": "South", "value": 30000, "count": 410},
  ...
]
</script>
```

`script.js` reads those rows, sums what the selected chip leaves, and writes the three tiles, the bars and the table. It is ordinary DOM code that draws once and again on each click: a page has no clock, so there is no timeline for it to keep in step with.

Live data, checked offline

The sales service is declared in `auditorium.toml`, next to a fixture file of the same shape:

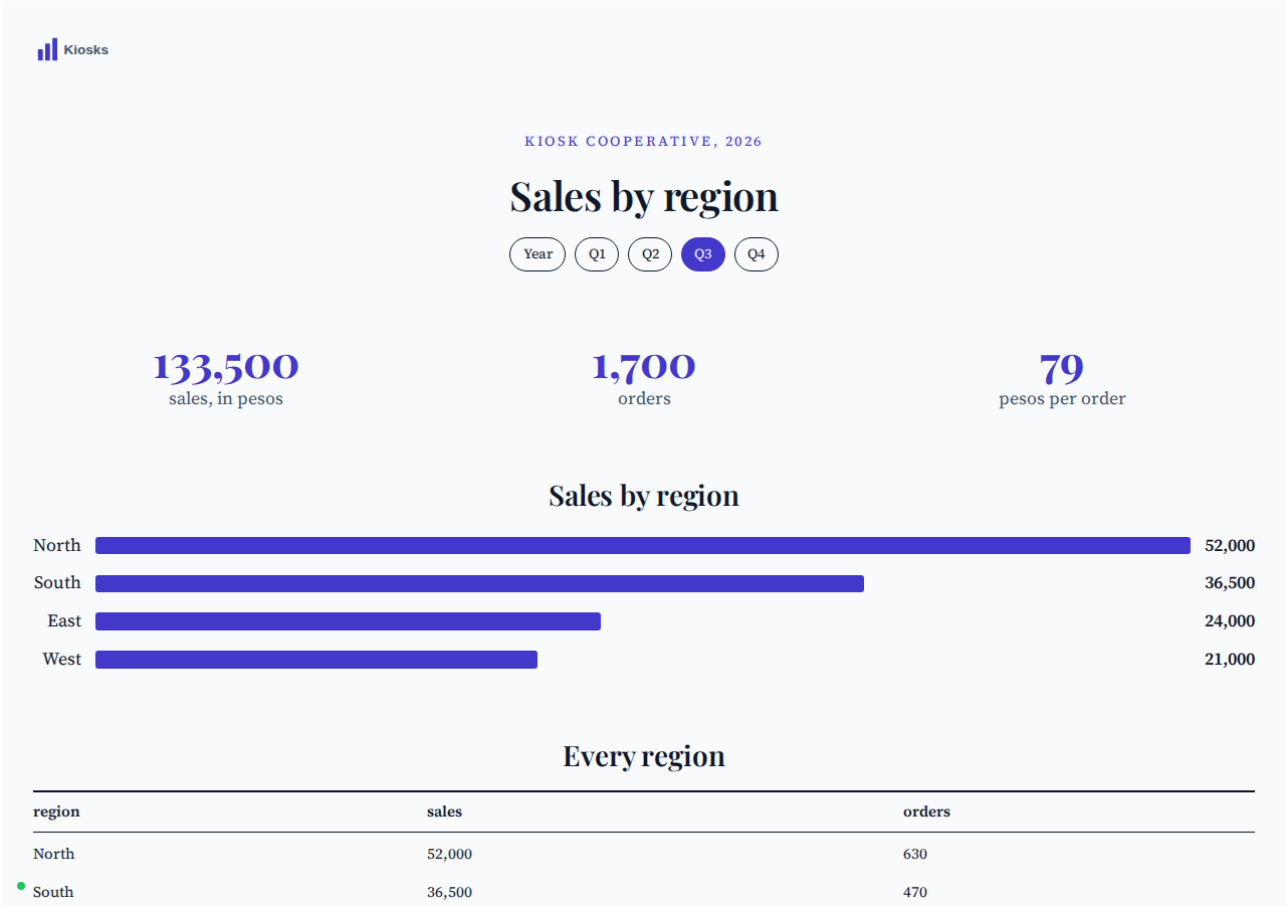
```
EX=$(python -c "import auditorium.templates as t; print(t.TEMPLATES)")
cp -r "$EX/dashboard/example" build/dashboard
grep -v '^#' build/dashboard/auditorium.toml
```

```
[data.sales]
url = "https://example.com/api/sales.json"
fixture = "assets/sales.json"
```

When the page opens in a browser, `script.js` asks the URL and redraws with the answer. When `check`, `render` or `export` open it, Auditorium answers that URL from the fixture, so a check never depends on a service being up and never touches the network.

```
auditorium check build/dashboard
```

```
0 errors, 0 warnings (structure and layout checked)
```



Ship it

```
auditorium bundle build/dashboard -o build/gallery/dashboard.html
```

```
✓ build/gallery/dashboard.html (font 0.16 MB, image 0.00 MB, other 0.00 MB, script 0.06
MB, style 0.25 MB)
340694 bytes in 1 file
```

The bundle carries the embedded rows, so the file works from disk; it still asks the declared URL when it opens and redraws if it answers. Open the dashboard and click the chips.

7

An explainer video

The job: a one-minute landscape video that explains how a heat pump warms a house, narrated, with each part of the figure appearing as the voice names it.

A clock instead of presses

An `explainer` has the same 1920 × 1080 stage as a deck, but it plays on a clock: each beat has a length, `data-d` in seconds, and the video is the beats in a row. Every surface seeks the same timeline, so the preview, a rendered frame and the exported HTML show the same thing at the same second.

```
auditorium template show explainer | head -3
```

```
explainer: a narrated landscape video, 1 to 6 minutes
A video that explains one idea out loud: a question, a statement, a figure, a comparison,
numbers, a closing line, each beat with its narration clip. Rendered to mp4.
formats: explainer (the first is the default)
```

Narration and cues

Each beat names its narration with `data-say`: an alignment JSON from a text-to-speech service or an aligner, which gives the time of every word, and the clip it describes. The voice starts 0.3 s into the beat.

```
EX=$(python -c "import auditorium.templates as t; print(t.TEMPLATES)")
cp -r "$EX/explainer/example" build/explainer
head -c 230 build/explainer/assets/vo/03.json; echo
```

```
{"audio": "03.wav", "words": [{"word": "A", "start": 0.05, "end": 0.225}, {"word":
"cold", "start": 0.295, "end": 0.635}, {"word": "liquid", "start": 0.705, "end": 1.155},
{"word": "takes", "start": 1.225, "end": 1.62}, {"word": "
```

With word times, an element can enter when the voice says a phrase instead of at a number of seconds. `data-cue` does that, so re-recording the narration at another pace moves the figure with it:

```
<section data-beat="figure" data-d="10" class="figure" data-say="assets/vo/03.json">
  <h2>Outside air in, heat up</h2>
  <figure>
    <svg viewBox="0 0 800 400">
      <rect x="40" y="120" width="200" height="160" rx="16" ... data-enter="draw"/>
      <path d="M 240 200 L 560 200" ... data-cue="compressor" data-enter="draw"/>
      <circle cx="400" cy="200" r="36" class="fill-accent" data-cue="compressor" data-
enter="pop"/>
      <rect x="560" y="120" ... data-cue="makes it hot" data-enter="draw"/>
    </svg>
  </figure>
</section>
```

The check holds a video to its voice: it fails a cue the narration never says (`cue-missing`), a beat shorter than its speech (`overrun`), and on an explainer or a short it warns when a beat is too short to read its text (`reading-time`) or when nothing moves and nobody speaks for too long (`dead-air`). The aurora layer behind the beats is there for that last rule: a moving background keeps a still beat from freezing the video.

```
auditorium check build/explainer
auditorium help rule dead-air | head -6
```

```
0 errors, 0 warnings (structure and layout checked)
# rule dead-air: A stretch where nothing moves and no voice plays for longer than the
format allows.

warning, layout pass. Explainers (4 s) and shorts (2 s) only: a deck waits for its
presenter.

Inside each beat, the check finds the time no animation runs (an entrance, exit, idle,
emphasis, camera or ambient) and no narration clip plays, then seeks it every 0.25 s and
keeps the stretch only if the frames do not change, so a `track()` function or a counter
counts as motion. A beat that holds still past the limit reads as a frozen video. Give
the stretch something that moves, or shorten the beat.
```

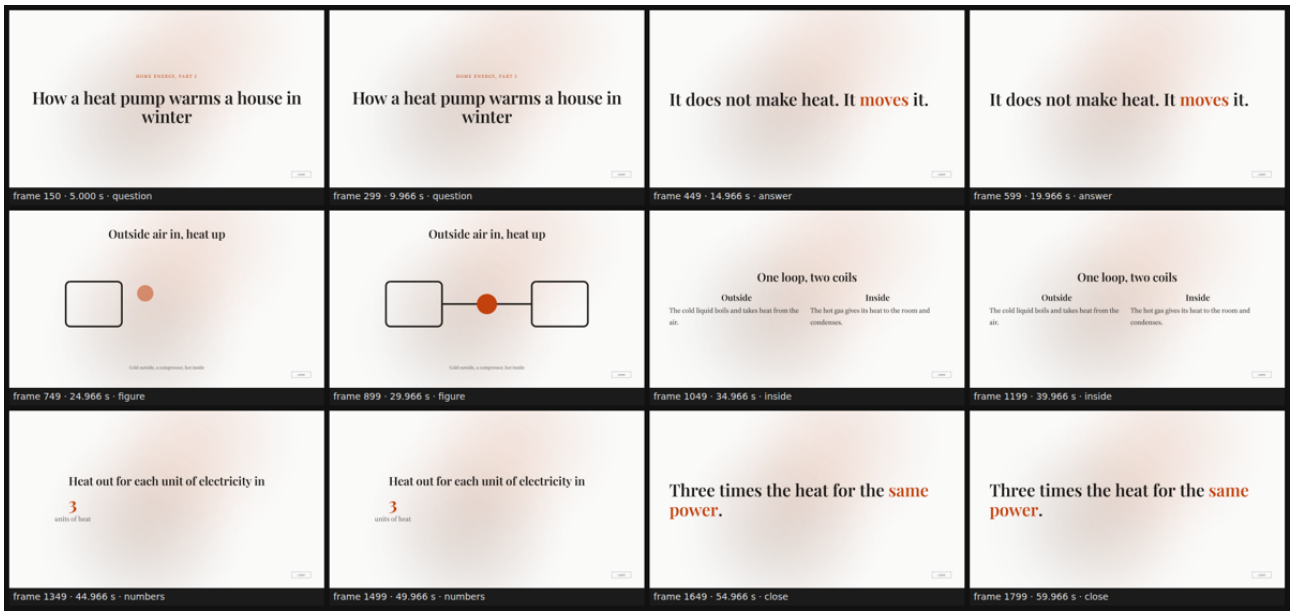
See the whole video at once

Watching a minute of video after every edit is slow. `sheet` lays the timeline out as one image, each beat at its end and frames between, each tile labelled with its frame number, time and beat:

```
auditorium sheet build/explainer --between 1 -o build/explainer-sheet.png | tail -3
```

```
frame   1649    54.966 s  close  (between)
frame   1799    59.966 s  close
```

```
12 tiles in build/explainer-sheet.png; auditorium render --from F --to F+1 reproduces
frame F
```



Render

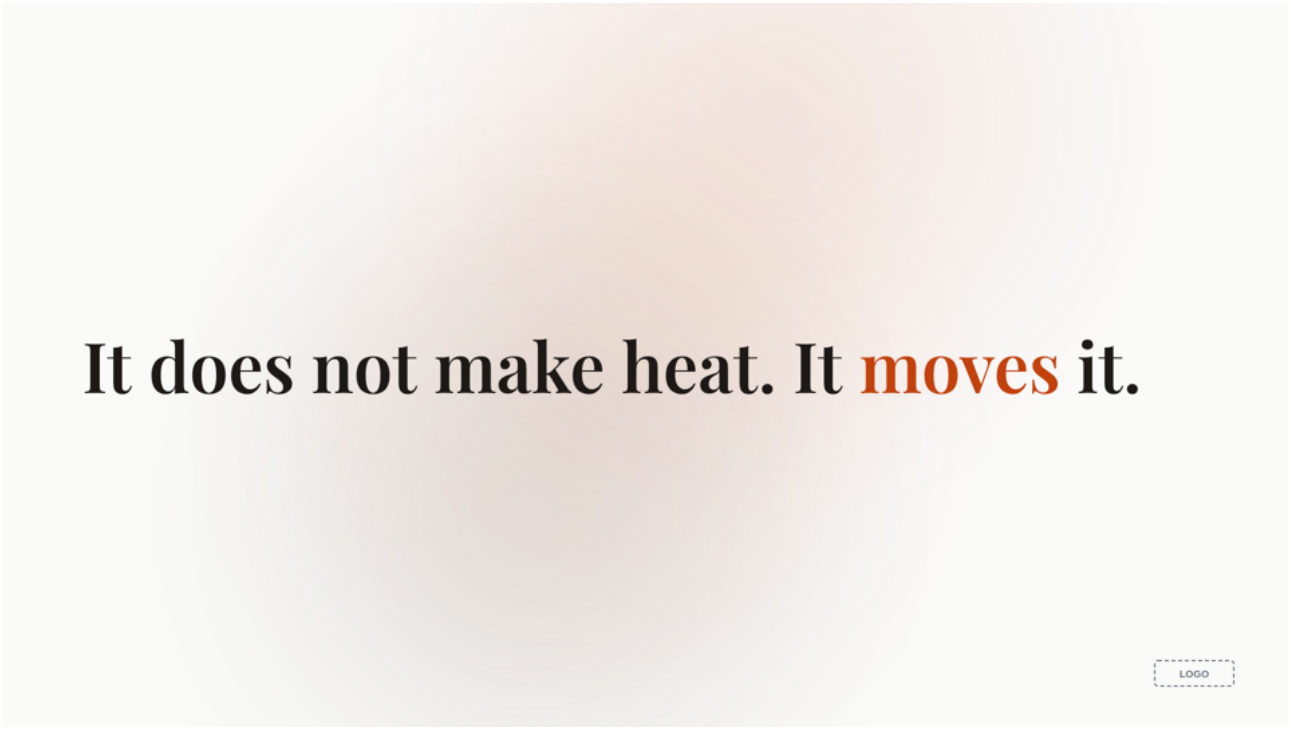
`render` steps Chromium through the timeline frame by frame, captures each frame, mixes every clip at its time, and encodes with ffmpeg. It is deterministic: two renders of one project are byte-identical. `--from` and `--to` render a range of frames, and `--jobs` splits the frames across several browsers.

```
auditorium render build/explainer --fps 30          # build/explainer/dist/explain-
er.mp4
auditorium render build/explainer --jobs 4          # four browsers at once
```

This book renders no video. It renders one frame, 18 s in, as a PNG sequence of length one, which is the same code path without the encoder:

```
auditorium render build/explainer --format png-sequence --from 540 --to 541 -o build/ex-
plainer-frames
```

```
✓ Rendered to build/explainer-frames
```



The HTML version

The bundle plays the same timeline in a browser, with its narration:

```
auditorium export build/explainer -f html -o build/gallery/explainer.html
```

```
✓ build/gallery/explainer.html (audio 0.27 MB, font 0.16 MB, image 0.00 MB, other 0.00
MB, script 0.06 MB, style 0.25 MB)
702822 bytes in 1 file
```

Open the explainer and press space: each press plays the next beat, with its voice.

8

A short video

The job: a vertical clip of under twenty seconds for Reels, TikTok and Shorts, telling people that a dripping tap wastes a bathtub of water a week, over music that drops when the voice speaks.

The vertical stage and its safe area

A `short` is 1080 × 1920 on a clock, 15 to 60 s long. The apps lay their buttons and captions over the top and bottom of the video, so the format has a safe area that every layout keeps clear, the widest band each platform documents, and one file posts to all of them. The check fails text that strays into it.

```
auditorium template show short | head -3
EX=$(python -c "import auditorium.templates as t; print(t.TEMPLATES)")
cp -r "$EX/short/example" build/short
```

```
short: a vertical video, 15 to 60 s
A clip for Reels, TikTok or Shorts: a hook, a number, a line, a call to action, over a
music bed that ducks under optional narration. Rendered to mp4.
formats: short (the first is the default)
```

A music bed that ducks

Music is an `<audio>` element like any other asset. `loop` repeats it to the end of the video, `data-gain` sets its level in the mix, and `data-duck` presses it down by about 12 dB while the narration speaks:

```
<section data-beat="hook" data-d="5" class="title" data-say="assets/vo/01.json">
  <span class="kicker">City water</span>
  <h1 data-enter="words">A dripping tap wastes water all day</h1>
  <audio src="assets/bed.wav" data-in="0" data-gain="0.5" data-duck loop></audio>
</section>
```

When you have no music, `auditorium music` composes a bed with Euterpe, in one of three moods. Without `--length` it writes one loop that tiles with no seam, which is what `<audio loop>` wants; with `--length` it lays loops end to end, quiet at the ends and full between, and fades them in and out. To use the result, write it into the project's `assets/` and point the `<audio>` at it; this book keeps the example's own bed.

```
auditorium music --mood bright --bpm 110 -o build/bright-bed.wav
```

```
build/bright-bed.wav: bright at 110 bpm, one seamless loop
1 loops of 8.72727 s: full
```

The check mixes the audio the way the render will and warns when the mix clips or the music is louder than the voice (`audio-peak`):

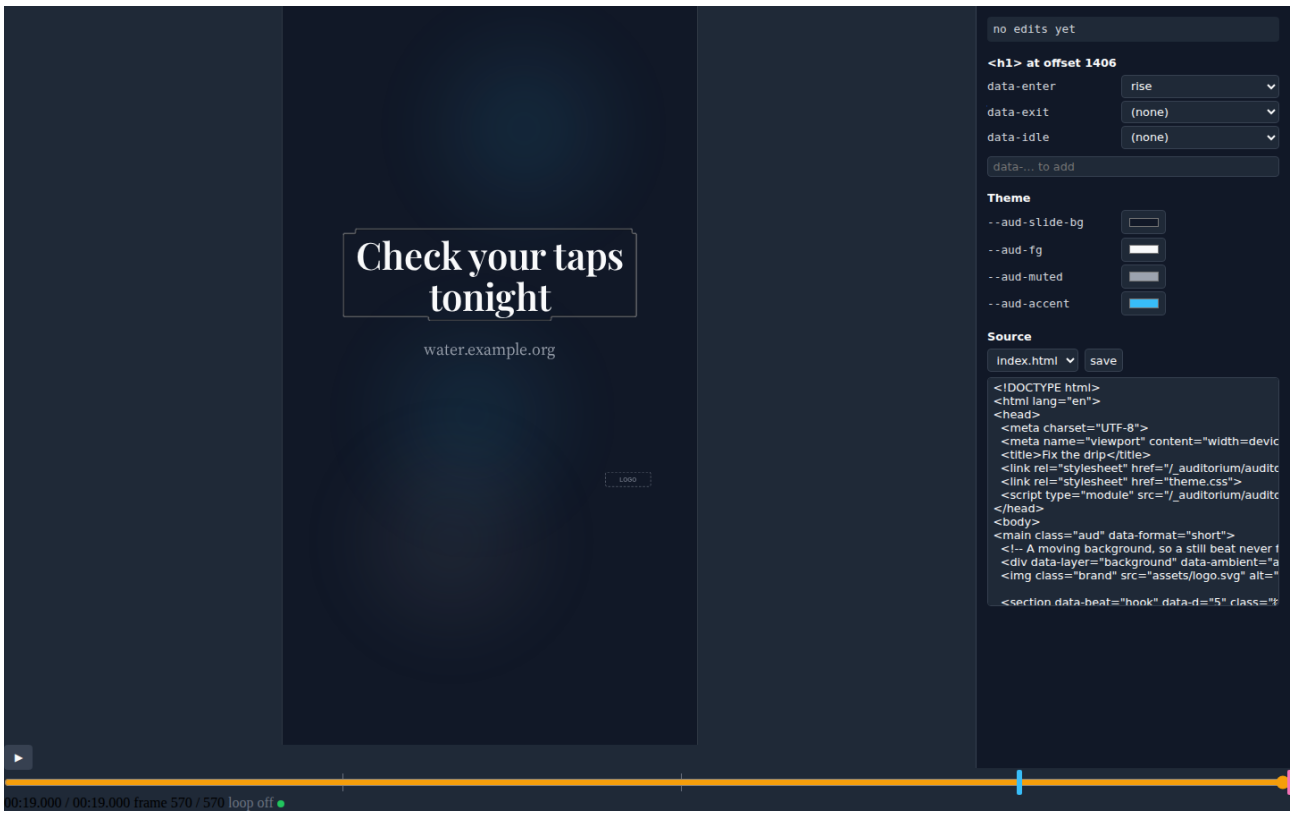
```
auditorium check build/short
```

```
0 errors, 0 warnings (structure and layout checked)
```

Edit in place

Some changes are faster to make by looking: a word in a headline, the moment an element enters, the accent colour. `preview --edit` adds an editing pane to the preview. Click a text on the stage to type over it; select an element to change its motions from lists of the valid names, and drag its `data-in` and `data-out` as handles on the scrubber; set the theme's variables with colour pickers; or edit `index.html`, `theme.css` and `script.js` in a source pane.

```
auditorium preview build/short --edit
```



Each change rewrites only the bytes it names in the file, so your formatting and comments survive, then reloads the stage and runs the structure pass of the check. The editor is off unless you pass `--edit`, and the server writes nothing outside the project folder.

Code that moves with time

A short sometimes needs motion no attribute describes: a needle, a curve that grows, a counter in a custom format. That code goes through `auditorium.track(fn, {beat, from, to})`, which calls `fn(seconds)` on every seek, and draws its randomness from `auditorium.rng(seed)`:

```
const needle = document.querySelector("#needle");
auditorium.track((s) => {
  needle.style.rotate = `${Math.min(s, 3) * 40}deg`;
}, { beat: "number", from: 0, to: 3 });
```

A function of time can be seeked, so the preview, the render and the export agree on every frame. A `requestAnimationFrame` loop or `Math.random` cannot, and the check flags both.

Ship it

The video itself is `auditorium render build/short`, an MP4 at 1080 × 1920 with the narration and the bed mixed in. The HTML version plays in a browser:

```
auditorium export build/short -f html -o build/gallery/short.html
```

```
✓ build/gallery/short.html (audio 0.13 MB, font 0.16 MB, image 0.00 MB, other 0.00 MB,
script 0.06 MB, style 0.25 MB)
513913 bytes in 1 file
```

Open the short and press space to play each beat.

9

Gallery

Every template's finished example, exported by this book's build with `auditorium export -f html`. Each is one self-contained file: the runtime, the styles, the fonts, the images and the audio are inside, and it loads nothing from the network. Open one, then use the right arrow or space to go forward, the left arrow to go back, `r` to restart and `f` for full screen. On a phone, tap the right side of the screen.

EXAMPLE	WHAT IT IS	MADE WITH
talk	a lecture on gradient descent: steps, notes, a figure that draws, math, code	`auditorium export build/talk -f html`
poster	a square announcement for an open lab day	`auditorium export build/poster -f html`
pitch	a solar cooperative asks to fund fifty schools: counting stats, a photo, partner logos	`auditorium bundle build/pitch`
report	water use in a district's schools: prose, an SVG chart, a table, bars; reflows on a phone	`auditorium bundle build/report`
dash-board	sales by region with quarter chips that filter the tiles, bars and table	`auditorium bundle build/dashboard`
explainer	a narrated minute on heat pumps, figure parts entering on spoken cues	`auditorium export build/explainer -f html`
short	a vertical clip on dripping taps, with narration over a ducking music bed	`auditorium export build/short -f html`
motion	the motion library: every entrance, exit, idle and em-phasis in one explainer	`auditorium export examples/motion -f html`
hello	the two-slide deck from the Quick Start	`auditorium bundle examples/hello`

`bundle` and `export -f html` are the same command; the chapters use both. Each chapter of the User Guide copies its template's example into `build/` before it exports, so the commands above are the ones those chapters ran. The motion gallery is a project in the repository:

```
auditorium export ../examples/motion -f html -o build/gallery/motion.html
```

```
✓ build/gallery/motion.html (font 0.16 MB, script 0.06 MB, style 0.25 MB)
338288 bytes in 1 file
```

The files this build wrote:

```
cd build/gallery && for f in *.html; do printf '%-16s %8d bytes\n' "$f" "$(wc -c <
"$f")"; done
```

```
dashboard.html     340694 bytes
explainer.html     702822 bytes
hello.html         333842 bytes
motion.html        338288 bytes
pitch.html         337432 bytes
poster.html        334578 bytes
report.html        337298 bytes
short.html         513913 bytes
talk.html         1104302 bytes
```

PART II

Developer Guide

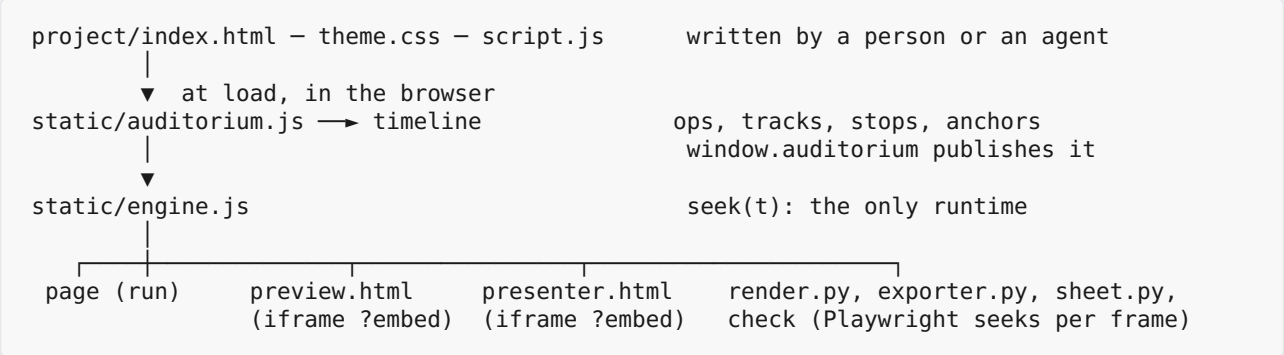
10

How Auditorium is built

This part is for someone who will change Auditorium: fix a surface, add a check rule, a motion, a template or a theme. It goes from the one idea down to the modules. The reasons behind each rule are in `DESIGN.md`, and the dated specs it cites are under `docs/design/`.

One timeline, one function

At load, `auditorium.js` walks the page once and builds a timeline: ops that put elements on and off the stage at an instant, and tracks that animate them over an interval. Nothing plays while it builds. Then `seek(t)` puts the page in the state it holds at time `t`, and every surface is a different clock driving that one function:



The reason is agreement. A second path that computed what the page shows would sooner or later disagree with the first, and the disagreement would show up as a render that does not look like the preview. So the preview and the presenter view embed the page itself in an iframe, and the Python tools drive the page in Chromium rather than modelling it.

Three rules keep `seek` exact. Every animation is paused and moved only by setting its `currentTime`, so a CSS transition (which drops out of `getAnimations()` when it ends) is banned in shipped and author CSS. A backward seek undoes every op and replays forward, because the same `t` reached by rewinding can differ from `t` reached fresh. And author code joins through `auditorium.track()` and `auditorium.rng()`, never a wall clock.

What the page publishes

The timeline never leaves the page. What the tools need, the page publishes as plain data on `window.auditorium`, with times in integer milliseconds. The renderer reads it through `published_timeline()`:

```
import asyncio
from pathlib import Path
from auditorium.render import published_timeline

published = asyncio.run(published_timeline(Path("build/dev/talk")))
print(published["format"], published["stage"], published["duration"], "ms")
for beat in published["beats"][:4]:
    print(beat["id"], "starts at", beat["t"], "ms, lasts", beat["d"], "ms, stops at",
beat["stops"])
print(published["stops"][:2])
```

```
deck {'w': 1920, 'h': 1080} 8272 ms
title starts at 0 ms, lasts 1300 ms, stops at [1300]
part-1 starts at 1301 ms, lasts 600 ms, stops at [1901]
claim starts at 1902 ms, lasts 850 ms, stops at [2752]
steps starts at 2753 ms, lasts 1804 ms, stops at [2754, 3355, 3956, 4557]
[{'t': 1300, 'hold_ms': 1500}, {'t': 1901, 'hold_ms': 1500}]
```

Frames from a timeline

A deck's stops take no time on the timeline: the presenter waits there. A rendered video must dwell on them, or it would rush past every reveal. `render_schedule()` is the pure function that maps output frames to timeline times, holding each stop for its `hold_ms`:

```
from auditorium.render import render_schedule

schedule = render_schedule(published["duration"], published["stops"], fps=30)
print(len(schedule), "frames")
print(schedule[:5], "...", schedule[-3:])
```

```
788 frames
[0, 33, 66, 100, 133] ... [8272, 8272, 8272]
```

The preview's frame counter reimplements this function in JavaScript, and a test asserts the two agree, so the frame number the preview shows is one you can pass to `render --from`.

The modules

MODULE	DOES
`cli.py`	the `auditorium` command, one Typer function per command
`server.py`	serves the folder, the runtime under `/_auditorium/` and the shells; watches files for hot reload; relays presenter commands; the `--edit` endpoints
`static/`	the runtime (`auditorium.js`, `engine.js`, `client.js`), the shells, `editor.js` and its stage half `stage-edit.js`, the CSS, vendored KaTeX, highlighter and fonts
`render.py`	frame schedule, Chromium capture (`beginFrame` on Linux), ffmpeg encode, `--jobs`
`mix.py`, `narration.py`	the audio mix graph, and alignment files for `data-say`
`exporter.py`, `sheet.py`	PNG stills and the PDF; the contact sheet
`bundle.py`	one HTML file: an HTML parser rewrites references into an asset table and an import map
`check/`	the rules, in passes: `structure.py` from the files, `layout.py` and `audit.js` in Chromium, `audio.py`, `reading.py`
`edit.py`	the byte-exact edits `preview --edit` makes
`assets.py`	`assets.yaml`: copies, captures, the image provider plugins, the cache
`music.py`	music beds with Euterpe
`relay.py`	the public relay
`templates/`, `themes/`, `help/`	the seven templates with their examples, the twelve themes, the help topics

The check

`check()` runs the passes cheapest first and stops after the first pass that finds an error, because a broken tag moves every box and measuring them would only add noise. `RULES` is the registry: a rule not in it cannot fire.

```
from collections import Counter
from pathlib import Path
from auditorium.check import RULES, check

print(Counter((r.group, r.level) for r in RULES.values()))
```

```
report = check(Path("build/dev/short"))
print(report.text())
```

```
Counter({('structure', 'error'): 11, ('layout', 'error'): 11, ('layout', 'warning'): 4})
0 errors, 0 warnings (structure and layout checked)
```

Every `Finding` names the file, the line, the rule, what was measured and the fix in the file's own terms, which is what lets a small model act on it without reading the source.

The server and the presenter

`create_app()` builds the FastAPI app that `run` and `preview` serve. The presenter broadcasts intent ("seek to t", "play from here to there"), never positions, and only the presenter's socket may drive; every surface runs the same deterministic engine, so a command is enough. Late tabs catch up from the last command.

```
from pathlib import Path
from auditorium.server import create_app

app = create_app(Path("build/dev/talk"), presenter_mode=True)
print(sorted(r.path for r in app.routes if r.path in ("/", "/preview", "/presenter", "/
ws", "/_auditorium")))
```

```
['/', '/_auditorium', '/presenter', '/preview', '/ws']
```

11

Extending Auditorium

Each recipe below says where the new piece goes, which registry lists it, and which test holds it. The gates are `make test`, `make lint` and `make docs`; CI runs them on every pull request, and the book's own gate, `make book-check`, builds this book with every block executed.

A check rule

A rule is a name in the registry, a function that yields findings, a fixture project that breaks it, and a help page that shows it broken and fixed. Every rule has all four:

```
from pathlib import Path
from auditorium.check import RULES

rules = set(RULES)
fixtures = {p.name for p in Path("../tests/rules").iterdir() if p.is_dir()}
pages = {p.stem for p in Path("../auditorium/help/rules").glob("*.md")}
print(len(rules), "rules;", "fixtures for all:", rules == fixtures, "; help pages for
all:", rules == pages)
```

```
26 rules; fixtures for all: True ; help pages for all: True
```

1. Add a `Rule(name, group, level, catches)` to `_RULES` in `auditorium/check/__init__.py`. `group` is the pass that runs it: `structure` reads the files and starts no browser; `layout` runs in Chromium.
2. Write the function. A structure rule takes the loaded `Project` and yields `finding(rule, file, line, message, fix)`; list it in `STRUCTURE` in `check/structure.py`. A layout rule measures in `check/audit.js` and is turned into findings in `check/layout.py`.
3. Add `tests/rules/<name>/index.html`, a project that breaks this rule; `tests/test_check.py` runs every fixture and expects its rule.
4. Write `auditorium/help/rules/<name>.md` with a broken and a fixed snippet; `auditorium help rule <name>` prints it, and the help tests run the snippets.

Write the message as what was measured against what, and the fix as an edit in the file's own terms. The reader is often a small model that will make exactly the edit you name.

A motion

Entrances, exits, idles, emphases, ambients and grades are tables in `auditorium/static/auditorium.js` (`ENTRANCES`, `IDLES`, `EMPHASES`, `AMBIENTS`, `GRADES`). The check reads the names from the same tables, so a new motion is valid as soon as it is in one:

```
from auditorium.check import vocab

print("entrances:", vocab.entrances())
print("idles:", vocab.idles())
print("layouts:", vocab.layouts())
```

```
entrances: ['fade', 'rise', 'pop', 'flip', 'grow', 'draw', 'words', 'count']
idles: ['float', 'tilt', 'pulse', 'ken-burns']
layouts: ['title', 'statement', 'cards', 'stats', 'columns', 'figure', 'logos', 'photo',
'table', 'chart']
```

The procedure, with its traps (opacity and transform as separate effects, DOM changes only at build), is `know-how/adding-a-motion.md` in the repository. A new motion also goes into `examples/motion`, and someone looks at its frames before it is done: a passing suite says it moves, not that it reads.

A template

A template is a folder under `auditorium/templates/`:

```
cd ../auditorium/templates/poster && find . -type f | sort
```

```
./assets/logo.svg
./example/assets/logo.svg
./example/index.html
./example/theme.css
./index.html
./template.toml
./theme.css
```

- `template.toml` holds `summary`, `for` and `formats`; `template list` reads it, and the first format is the default.
- `index.html`, `theme.css` and `assets/` are what `template new` copies, with every element to replace marked `data-placeholder`.
- `example/` is the same project finished. It stays in the package, and the tests hold it to the check: the skeleton must fail on its placeholders only, and the example must pass with no errors.

`tests/test_templates.py` covers each of those, and the themes test checks every bundled theme for contrast on your example.

A theme

A theme is a `:root` block of variables in `auditorium/themes/<name>.css`, and nothing else: a selector would reach into markup the templates and layouts own, and break when they change. `tests/test_themes.py` fails a theme that has a selector, misses the geometry palette, or fails WCAG AA on any template's example. A theme that fails contrast on one example is dropped, not patched for it.

```
sed -n '1,12p' ../auditorium/themes/dark.css
```

```
/* dark: near-black stage on a warm-black letterbox, off-white text, amber
   accent, and code colours for a dark panel. */
:root {
    --aud-letterbox-bg: #07080a;
    --aud-slide-bg: #14171c;
    --aud-fg: #e8e6e3;
    --aud-muted: #9a958d;
    --aud-accent: #f7c873;
    --aud-rule: #2a2e36;
    --aud-code-bg: #1c2026;
    --aud-code-keyword: #ff7b72;
    --aud-code-literal: #79c0ff;
```

An image provider

`generate:` entries in `assets.yaml` ask an image provider, a plugin registered under the `auditorium.image_providers` entry point group:

```
[project.entry-points."auditorium.image_providers"]
flat = "my_package:generate"
```

The function takes the prompt, the size, the seed and the format, and returns the image's bytes. This one paints a flat gradient; the block registers it by hand, where an installed package would register it through the entry point:

```
import io
from pathlib import Path
from PIL import Image
import auditorium.assets as assets

def generate(prompt, *, size, seed, format):
    image = Image.linear_gradient("L").resize(size).convert("RGB")
    out = io.BytesIO()
    image.save(out, format="PNG" if format == "png" else "JPEG")
```

```
    return out.getvalue()

assets.providers = lambda: {"flat": generate}

project = Path("build/ext/hero")
(project / "assets").mkdir(parents=True, exist_ok=True)
(project / "index.html").write_text('<main class="aud" data-format="square"><section
data-beat="a"><img src="assets/hero.png"></section></main>')
(project / "assets.yaml").write_text("assets/hero.png: {generate: a soft gradient, size:
[640, 360], seed: 1, provider: flat}\n")
for outcome in assets.resolve(project):
    print(outcome.as_dict())
print(Image.open(project / "assets/hero.png").size)
```

```
{'key': 'assets/hero.png', 'kind': 'generate', 'status': 'resolved', 'detail': ''}
(640, 360)
```

Use a generated image for a backdrop or an illustration no file gives you, never for a logo, a chart or text: a diffusion model draws something like it, not it.

The book

This book lives in `docs/`: `docs/scriptorium.yaml` lists the chapters in `docs/chapters/`, and `docs/shots.py` takes the screenshots. `make book` builds the site and the PDF, with the gallery, into `docs/site/`; `make book-check` also fails on any block that failed, on a file under `docs/examples/` no chapter tangles, and on a CLI command the book never names. The examples in `docs/examples/` are tangled from the chapters: edit the chapter, never the file.