scholatex

Print-ready teaching worksheets, the easy way

A small tag-based language for handouts and exercises,
built on LuaLaTeX

Gérard Dubard

Version 2.5

Released under the GNU General Public License v3.0

Contents

I Getting started **2**

1 Introduction **2**

2 The one rule **2**

3 Class options **3**

II Text and structure **3**

1 Text attributes **3**

2 Aliases and macros — the factoring tool **4**

3 Lists **5**

4 Escapes **5**

III Containers and layout **6**

1 Tables **6**

2 Images **6**

3 Boxes **7**

4 Grid (named-area layout) **7**

5 Drawing figures **7**

IV Maths **12**

1 Basics **12**

2 Algebra **14**

3 Analysis **16**

4 Probability **21**

5 Statistics 22

V Programming the document 22

1 Block aliases 22

2 Control flow 23

VI Reference 23

1 Security 23

2 Examples 23

3 Diagnostics 24

Part I Getting started

1 Introduction

scholatex is a small markup language for producing print-ready teaching materials — worksheets, exercise sheets, handouts, short assessments — without writing LaTeX. It is not a general-purpose replacement for LaTeX; it is a focused, consistent little language for the documents a teacher actually makes: a few framed exercises, a table of results, a couple of simple formulas, an image or two, laid out cleanly on a page meant to be printed and handed out.

It compiles to LuaLaTeX, so the output has full LaTeX typesetting quality, but the syntax is meant to be read and edited in minutes by someone who does not know LaTeX: no `\begin{tabular}{|c|c|}`, no counting ampersands, no hunting for the package that draws a coloured box. A single tag syntax covers text, tables, images, simple maths, framed boxes, lists and full-page named-area layouts — the building blocks of a worksheet. scholatex is a *closed language*: the backslash is an ordinary character, so raw LaTeX commands in the body are typeset literally rather than executed. Everything a worksheet needs is expressed through tags, which keeps documents consistent and safe to share.

It is built for the people LaTeX usually loses: teachers preparing classroom materials, and anyone who wants a clean printable sheet without climbing the LaTeX learning curve.

You write a `.tex` file with a tiny, readable syntax between `\begin{document}` and `\end{document}`. At compile time the `scholatex` class reads that body, transpiles it to LaTeX, and typesets it. Nothing to install beyond a LuaLaTeX distribution.

```
% !TeX program = lualatex
\documentclass[margins=20, size=12, imgdir=IMG]{scholatex}
\begin{document}

<Navy b 18pt c>My first scholatex document

This is a normal paragraph. <b>{Bold}, <i>{italic}, <Red>{red}.

<box line:Navy fill:AliceBlue radius:3 title:{A framed note}>{
Boxes, tables, images and maths all use the same tag syntax.
}

\end{document}
```

2 The one rule

Everything is a **tag**: `<attributes>` followed either by `{content}` (inline) or by a line that ends in `{` (a block). Attribute words follow a single case convention:

Form	Meaning	Examples
`lowercase`	a short keyword (style, alignment, skip)	`b`, `i`, `c`, `2tab`
`CamelCase`	a CSS colour (151 of them)	`Navy`, `SteelBlue`, `Crimson`
`UPPERCASE`	a font name	`DEJAVU SANS`

The order of words inside a tag never matters — emission is always normalised (page break, then vertical skips, then alignment, then tabs, then styles).

3 Class options

Set in `\documentclass[...]{scholatex}`:

Option	Default	Meaning
`margins`	`20`	`N` (all sides) or `{top,right,bottom,left}` in mm
`font`	`Latin Modern Roman`	main text font
`mathfont`	`Latin Modern Math`	math font
`size`	`11`	base font size in pt
`imgdir`	`img`	folder(s) searched for bare image names; accepts a comma-separated list, e.g. `{IMG, IMAGES/PNG}`
`tabwidth`	`8`	width of one tab, in mm
`lineheight`	`8`	height of one explicit line skip (`<line>`, `<Nlines>`), in mm
`linespread`	`1.0`	overall line spacing (`1.5` = one-and-a-half); tall maths never touches even at `1.0`
`scriptscale`	`100`	scale (%) of `up`/`down` scripts
`padding`	`2`	inner padding (mm) between a box/grid-area frame and its content
`lang`	`fr`	decimal separator: `fr` (comma) or `en` (point); affects both literal decimals in maths and interpolated values
`untrusted`	`false`	when `true`, runs the document's Lua in a restricted sandbox (see §1)

Headings carry no extra vertical space of their own, and no built-in colour: to style a heading (colour, weight, size, or a line skip before it), fold a heading keyword into an alias and use it (see §2).

Part II Text and structure

1 Text attributes

Inline styles — `b` bold, `i` italic, `u` underline, `emph`, `sf` sans-serif, `sc` small caps.

Colours — a colour is always one of the 151 CSS / svgnames colours, written in CamelCase: `Navy`, `Red`, `Tomato`, `SteelBlue`, `ForestGreen`, ... There is a single rule — no lowercase shortcuts — so the everyday colours are simply capitalised: `Red`, `Blue`, `Green`, `Gray`. These are exactly

the colours xcolor provides under `svgnames`: the 147 CSS Color Module names plus the four X11 extras `LightGoldenrod`, `LightSlateBlue`, `NavyBlue` and `VioletRed`. Names are case-sensitive — write `Seashell`, not `SeaShell`. The `README` groups all 151 by family for browsing.

Fonts and sizes — a font name in CAPITALS (`<DEJAVU SANS>{...}`); sizes as `Npt` or `Npx` (`<14pt>{...}`).

Alignment — `l` left, `c` centre, `r` right, `j` justified (the same letters as table columns).

Tabs and skips (the number is always a prefix) — `Ntab` indents the first line by N tabs; vertical skips follow singular/plural agreement: `line` or `1line` skips one line, `2lines`, `3lines`, ... skip several. Bare `tab` = `1tab`.

Scripts — `upN` raises text by N mm (superscript-like), `downN` lowers it (subscript-like): `x<up4>{2}`, `H<down2>{2}O`.

Page break — `nextpage`.

Section headings — `section`, `subsection`, `subsubsection` give the three heading levels, numbered automatically:

```
<section>First topic
<subsection>A detail
<subsubsection>A finer point
```

renders as '1 First topic', '1.1 A detail', '1.1.1 A finer point'. By default the numbers are hierarchical; a counter keyword on a heading sets the style of that level on its own, with no inherited prefix: `num` (1, 2, 3), `roman` (i, ii), `ROMAN` (I, II), `alpha` (a, b), `ALPHA` (A, B). Thus `<section ROMAN>` with `<subsection num>` gives `I`, `II` and, under each, `1`, `2` — not `I.1`. A heading can also be written as a **block** carrying a `title:` option, with its body in braces:

```
<subsection title:{A heading}>{
<tab>This body paragraph is indented on its first line.
}
```

A **table of contents** is printed with `<tableofcontents>`; give it a title in braces:

```
<tableofcontents>{Table of contents}
```

Combine anything in one tag:

```
<nextpage 2lines 2tab j Navy b>{A new page, a two-line skip, a two-tab
indent, justified, Navy, bold --- all before the first letter of the text.}
```

2 Aliases and macros — the factoring tool

This is what makes scholatex scale. Define a style **once**, at the top of the document, then name it everywhere instead of repeating its attributes. One edit at the definition restyles the whole document.

```
let title = <Navy b 18pt c>          % style alias
let h1    = <Navy b section>         % a heading style, reusable
let p     = <tab>                    % a standard indented paragraph
<title>My heading
<h1>First topic
<p>{ A paragraph, indented and justified, named not described. }

let n = 7                            % value, usable in #{...}
Seven squared is #{n*n}.

let greet{name} = Hello #name!       % text macro with parameters
```

Change `let h1 = <Navy b section>` to `<ForestGreen b section>` once and **every** first-level heading follows — the single point of control that keeps a long document consistent.

3 Lists

`<list:STYLE>{ ...}` makes a list; the style is required and comes right after the name. One item per non-empty line — no item tag. A `<list:...>` written under an item becomes its sub-list, nested as deep as you like.

```
<list:disc>{
Fruits
<list:circle>{
apples
pears
}
Vegetables
}
```

Styles — bullets: `none disc circle square`; numbered: `decimal alpha ALPHA roman ROMAN` (the case of the keyword sets the case of the letters); checkboxes: `check`.

4 Escapes

To print a character that scholatex treats specially, double it: `« »` give a literal `< >`, `{{ }}` a literal `{ }`, and `##` a literal `#`. A backslash `\` is an ordinary character — write it as itself, so a Windows path like `C:\Users` just works. The characters `_ & % ~ ^` are escaped automatically.

Braces carry structure (they delimit blocks and groups), so a *literal* brace must be **balanced**: write the pair `{{...}}` to print `{...}`. A lone, unmatched `{{` or `}}` is reported as an unbalanced brace naming its line, rather than silently swallowing the rest of the document — so set-builder notation like `{{ x :  x > 0 }}` is written as a pair and prints as one. Angle brackets and hashes have no such constraint; only braces need to balance.

A line break is the tag `<nextline>`; it works in running text and inside table cells. In ordinary text a blank line already starts a new paragraph, so `<nextline>` is only needed for a break *without* a paragraph change.

A line whose first non-space character is `%` is a **comment** and is dropped. A bare `#` not followed by a name or `{...}` is a literal `#`: only `#name` and `#{expr}` interpolate.

Part III Containers and layout

1 Tables

Columns are declared in brackets, **one two-letter placement code per column**. The first letter is vertical (`t` top, `m` middle, `b` bottom), the second horizontal (`l`, `c`, `r`). So `mc` is middle-centre, `br` bottom-right. Columns share the page equally; `N:` before the code fixes a width in mm (`[40:tl, 30:mc]`). Cells are separated by `|`, rows by line breaks, and `<nextline>` breaks a line inside a cell.

```
<table [mc, tl, br] borders header>{
Figure | Formula | Value
<img 25>{cat.png} | $1/2 + 1/3$ | $5/6$
}
```

`borders` draws rules; `header` bolds the first row. `gap:N` sets the horizontal spacing between columns, in mm.

A cell may span several columns with `colspan:N`, or several rows with `rowspan:N` ($N \geq 2$). For a `rowspan`, each cell it covers on the lines below is marked with a lone `..`

```
<table [mc, ml, mc, mc] borders header gap:3>{
<colspan:4 mc>{Term report}
Day | Subject | Mark | Coef.
<rowspan:2 mc>{Monday} | Maths | 15 | 4
. | French | 12 | 3
}
```

Colour options act **inside** the table, on its cells and rules — `fill:` colours every body cell, `line:` the rules, `text:` the text colour, and `headerfill:` / `headertext:` the header row. These are not box options: a table has no `radius:`, `title:` or outer frame. To frame a table, wrap it in a `<box>`.

2 Images

```
<img>{chat.png}        % fills the available width (the cell or line)
<img 25>{chat.png}     % 25 mm wide
<img 25x15>{chat.png}  % fits a 25x15 mm box, ratio preserved
```

A bare name is searched in each folder of `imgdir` in turn, then at the project root. An explicit path always works, with or without `./`.

3 Boxes

```
<box line:Crimson fill:MistyRose radius:4 title:{A note}>{
Content here.
}
```

Options: `line:` frame colour, `fill:` background, `text:` text colour, `radius:N` rounded corners (mm), `width:N` or `width:N%`, `boxrule:N`, `sep:N` (inner padding, mm; `boxsep:N` is a synonym), `break:yes` (allow page breaks), `title:{...}`, `titlefill:`, `titletext:`. A line containing only `--` splits a box into an upper and a lower region.

`<row gap:N>{ ...}` lays its child boxes side by side, with equal widths and equalised heights. A `row` accepts only `<box>` children.

A box also takes a **two-letter placement code** (`tl tc tr ml mc mr bl bc br`, default `tl`) that positions its content; the vertical part needs a `height:` to have room to act.

4 Grid (named-area layout)

For full-page layouts, `<grid>` borrows CSS Grid's named-area idea. A `template:[ ...]` of quoted rows draws the layout; each word names the cell at that position. A name repeated **horizontally** spans columns; repeated **vertically** it spans rows. A dot `.` is an empty cell. Each name must form a solid rectangle.

```
<grid template:[
  "title  title  logo"
  "intro  info   logo"
  "body   body   body"
] gap:4>{
  <area title>{ <Red b 16pt>Maths assessment }
  <area logo >{ <img>{blason.png} }
  <area intro>{ Instructions: no calculator. }
  <area info >{ Name: <nextline> First name: }
  <area body line:Navy fill:AliceBlue radius:2 title:{Exercise 1}>{
    Solve the equation...
  }
}
```

An area can be framed like a box by giving it the same options (`line:`, `fill:`, `radius:`, `title:`). An area with no options stays invisible (a bare cell). The grid itself takes `width:` (a percentage of the text width or a millimetre value) and `height:` (a millimetre value fixing the total height).

5 Drawing figures

`<draw>` draws a plane geometric figure from a description alone: you name the shape and give its measurements, and the coordinates are computed trigonometrically and emitted as hard numbers. One drawing unit is one centimetre, so `side:5` draws a 5 cm side — nothing is rescaled, and a figure wider than the page is simply the cue to adjust the measurements before printing.

```
<draw>triangle ABC equilateral side:5 marks:on measures:cm
```

`marks:on` codes the equal sides and right angles; `measures:cm` or `measures:mm` labels each side with its length, set parallel to the side. Both are off by default. The block form `<draw>{ ...}` holds several figures (see *Composite figures* below); the inline form above is a single figure.

The figures

You write	You get
`triangle ABC equilateral side:s`	equilateral triangle
`triangle ABC isosceles side:e base:b`	isosceles, equal sides `e`, base `b`
`triangle ABC right sides:(p,q)`	right angle at the first point (or `right:X`), legs `p`, `q`
`triangle ABC sides:(a,b,c)`	three sides
`triangle ABC AB:3 BC:4 CA:5`	three sides named one by one (same as `sides:(3,4,5)`)
`triangle ABC sides:(a,b) angle:t`	two sides and the included angle
`triangle ABC angles:(A,B) side:c`	two angles and a side
`square ABCD side:s`	square
`rectangle ABCD sides:(w,h)`	rectangle
`rhombus ABCD side:s angle:t`	rhombus
`parallelogram ABCD sides:(a,b) angle:t`	parallelogram
`trapezoid ABCD bases:(b1,b2) height:h`	trapezoid
`kite ABCD sides:(a,b) angle:t`	kite: upper pair `a`, lower pair `b`, apex angle `t`
`pentagon`, `hexagon`, `octagon`	regular polygons by name
`polygon ABCDE side:s`	regular polygon, any number of points
`circle O radius:r` `circle O diameter:d`	circle by centre and length
`circle O radius:AB`	radius given by a placed segment (compass span)
`circle ABC`	circle through three placed points (circumscribed)
`circle ABC inscribed`	incircle of the triangle `ABC`

A `kite` is distinct from a `rhombus`: it has two pairs of adjacent equal sides of *different* lengths, the rhombus being the special case where both pairs are equal.

When a figure has two pairs of equal sides of *different* lengths — a kite, a rectangle, a parallelogram — `marks:on` tells the pairs apart by the number of ticks: a single tick on the one pair, a double tick on the other. A kite therefore never reads as a rhombus, nor a rectangle as a square. On a composite figure a shared edge is ticked once, not once per figure, so the coding stays clean where two pieces meet.

Composite figures: sharing an edge

Several figures in one `<draw>{ ...}` block share their points. When a figure repeats a point already placed, it is grafted onto it; when two figures share an **edge** (two points), the new figure deduces that edge's length from the existing one — so only the first figure states its side — and is flipped to the far side of the edge so the pieces abut instead of overlapping.

```
<draw>{
  rhombus ABEF side:4 angle:60
  square ABCD
  triangle BCG equilateral
  triangle ADK right:D DK:6 measures:cm
}
```

Point names, loops, rotation

Point names are single letters written glued together — `triangle ABC` names A, B and C. When a figure needs many points, or names with a digit, they are written as a parenthesised, comma-separated list: `triangle (O, A0, B0)`. The rule is one of the first character: an opening parenthesis introduces the list form, anything else is the glued form.

A draw block runs the same loops as the rest of the language, `for VARIABLE in FROM..TO { ...}`. Inside a loop, `#k` inserts the loop variable and `#{expr}` evaluates an expression, so point names and measurements can be generated. The attribute `rotate:`θ turns a figure by θ degrees about its first point, and `labels:off` hides the vertex names. On the tag itself, `scale:F` multiplies every coordinate of the picture by `F` while leaving the text size alone (default `1`; it applies to every form of `<draw>` — figures, axes, trigcircle, solids). Twelve isosceles triangles sharing an apex, stepped 30° apart, fan into a full turn:

```
<draw>{
  for k in 0..11 {
    triangle (O, A#k, B#k) isosceles side:4 base:2 rotate:#{k*30} labels:off
  }
}
```

Points and segments by coordinate

Two low-level primitives sit below the named figures. A point is placed at explicit coordinates, a segment is drawn between two points. Coordinates are declared once with `let NAME = {x, y}`, then referred to by name. A point's label is placed clear of any segment leaving it.

```
let P0 = {-1.6, -2.3}
let P1 = {-1, 1.5}
let P2 = {2.7, 3.1}
<draw>{
  point(P0)
  point(P1)
```

```
  point(P2)
  line(P1, P2)
}
```

An argument may also be a literal pair, `line({0,0}, {3,2})`, and a segment takes `measures:` like a figure side. Because each coordinate is an ordinary expression, a loop can compute endpoints — twelve rays around a disc make a sun:

```
<draw>{
  circle O radius:1.2
  for k in 0..11 {
    line({1.2*math.cos(k*math.pi/6), 1.2*math.sin(k*math.pi/6)},
         {2.4*math.cos(k*math.pi/6), 2.4*math.sin(k*math.pi/6)})
  }
}
```

In a coordinate frame: `axes:`

The figures above are *constructed* — you give measurements and the coordinates are computed. The opposite need, a graduated frame on which points are placed *at* their coordinates, is the same block in analytic mode. `<draw axes:{xmin,xmax,ymin,ymax}>` draws the axes and switches every line to the analytic vocabulary; without `axes:` the block stays in the construction mode above. One unit is still one centimetre, and the drawing is clipped to the declared window.

```
<draw axes:{-5,5,-5,5}>{
  point A (2,3)
  point B (-1,1)
  vector u (3,1)
  line y = 2x-1
  line through A B
}
```

Each directive takes coordinates, never measurements. `point NAME (x,y)` places and labels a point; a placed name is then available to later lines. `vector u (x,y)` draws a free vector from the origin, `vector AB` the bound vector between two placed points. `line` accepts an equation (`y = 2x-1`, `x = -3`, or the general `2x+3y = 6`) or `line through A B`.

A named vector is remembered and composes. `vector s = u+v` (or `u-v`) draws the sum or difference, labelled with the expression itself, written along the arrow. `from:` moves any vector's tail — to a placed point, a literal pair, or `~u`, the *tip* of a drawn vector — and `style:` restyles the copy (`style:dashed-gray`, `style:Blue`). The parallelogram and triangle rules then read on one frame: each side is copied dashed to the far tip, the sum closes the parallelogram, and the difference runs from the tip of `v` to the tip of `u`:

```
<draw axes:{-5,5,-5,5}>{
  vector u (3,1)
  vector v (1,3)
```

```
  vector v from:~u style:dashed-gray
  vector u from:~v style:dashed-gray
  vector s = u+v style:Blue
  vector d = u-v from:~v style:Red
}
```

A circle carries a tangent at a point on it. `circle` takes `center:` (a placed point or a literal pair) with `radius:`, or an equation `x^2+y^2 = 4`; a named circle (`circle C ...`) then accepts a `tangent to:C at:P`:

```
<draw axes:{-5,5,-5,5}>{
  point T (0,3)
  circle C center:(0,0) radius:3
  tangent to:C at:T
}
```

`region` shades the half-plane of a linear inequality (`<`, `>`), the boundary drawn dashed:

```
<draw axes:{-5,5,-5,5}>{
  region y > x+1
}
```

The trigonometric circle.

One synthetic line draws the circle, its axes, and the sixteen remarkable angles labelled by their principal measure in $(-\pi, \pi]$:

```
<draw>trigcircle radius:4
<draw>trigcircle radius:4 values:on
<draw>trigcircle radius:4 step:{pi/6}
<draw>trigcircle radius:4 range:{0, pi/2} step:{pi/12} values:on
```

`radius:` is in centimetres (default 4). `step:` replaces the remarkable set by the multiples of a fraction of π; `range:` restricts to an arc — the first-quadrant close-up. `values:on` adds the dashed cosine and sine projections of the drawn first-quadrant angles, exact ($\frac{1}{2}$, $\frac{\sqrt{2}}{2}$, $\frac{\sqrt{3}}{2}$) at the remarkable ones, two decimals elsewhere. Labels too crowded to read are a compilation error suggesting a larger step.

Solids in cabinet projection.

`solid` lines draw the space-geometry repertoire in the French *perspective cavalière*: receding axis at 45°, ratio 0.5 (override with `angle:` and `ratio:`). Hidden edges are *computed*, not declared: a face of a convex solid is visible exactly when the projection preserves its outward orientation, and an edge is dashed exactly when both of its faces are hidden. Several solids in one block sit side by side.

```
<draw>{
  solid cube ABCDEFGH edge:4
```

```
  solid pyramid SABCD base:4 height:5
}
<draw>{
  solid cuboid sides:{5, 3, 2}
  solid cylinder radius:1.5 height:4
  solid cone radius:2 height:4.5
  solid sphere radius:2.5
}
```

Vertex names are optional on `cube`, `cuboid` (eight letters, base then top) and `pyramid` (apex first). Plane sections are not in this release.

Part IV Maths

1 Basics

Wrap maths in `$...$`. A small mini-language keeps it light:

You write	You get
`*`	$\times$
`+-`	$\pm$
`<= >= !=`	$\leq \quad \geq \quad \neq$
`a/b`	fraction (chained `a/b/c` reads as `(a/b)/c`)
`x^2  x_i`	power / index (bind tighter than `/`)
`sqrt(2)`	$\sqrt{2}$
`sum(i=1, n) i`	$\sum$ with bounds, display style
`prod(k=1, n) k`	$\prod$ with bounds, display style
`int(x) f(x)`	$\int f(x)\,\mathrm{d}x$ (differential added)
`int(x=a, b) f(x)`	$\int_a^b f(x)\,\mathrm{d}x$
`contourint(C) f(z)`	$\oint$ contour integral
`pvint(x=a, b) f(x)`	p.v.$\int$ principal value
`meanint(x=a, b) f(x)`	$\fint$ average integral
`dy/dx`, `df/dx`	derivative, upright d (ISO)
`sin(x) cos(x) ln(x)`	upright function names
`abs(x)`	$\lvert x\rvert$
`norm(v)`	$\lVert v\rVert$
`vec(AB)`	$\overrightarrow{AB}$
`floor(x) ceil(x)`	$\lfloor x\rfloor \quad \lceil x\rceil$
`bar(z)`, `conj(z)`	$\overline{z}$ (conjugate, mean, closure)
`not(A cup B)`	$\overline{A \cup B}$ (logical negation)
`lim(x->0) f(x)`	limit, target under the word
`partial`, `nabla`	∂, ∇
`pi`, `alpha`, ...	Greek letters
`inf`	∞

The helpers nest, so the secondary-school staples come for free: `norm(vec(AB))` is the norm of a vector, `vec(AB) + vec(BC) = vec(AC)` is Chasles' relation.

Sets, logic and relations

A worksheet from collège to the upper years needs more than arithmetic. The mini-language carries the standard notation as plain words and ASCII shortcuts, so a statement reads as one types it.

Number sets — a doubled capital gives the blackboard-bold set, the ASCIIMath convention. Doubling avoids any clash with a one-letter variable: `NN` $\mathbb{N}$, `ZZ` $\mathbb{Z}$, `DD` $\mathbb{D}$, `QQ` $\mathbb{Q}$, `RR` $\mathbb{R}$, `CC` $\mathbb{C}$, plus `PP KK HH FF EE UU` for the advanced ones.

Comparisons — inside a formula, `< > <= >= !=` give $< > \leq \geq \neq$. (In running prose, double the chevrons — `«=»` prints a literal $\Leftrightarrow$.)

Quantifiers and logic — `forall` $\forall$, `exists` $\exists$; the connectives `and` $\wedge$, `or` $\vee$, `lnot` $\neg$, `neg` $\neg$. Implication and equivalence come both as words and as arrows: `=>` or `implies` $\Rightarrow$, `«=»` or `iff`

⇔. The longer arrow forms always win over the comparison they contain, so `abs(x) <= 1 <=> -1 <= x <= 1` reads correctly inside a formula.

Negation — a single rule: `!` in front of a relation or quantifier negates it, choosing the proper struck-through glyph. So `!in` is $\notin$, `!exists` is $\nexists$, `!subset` is $\not\subset$, `!subseteq` is $\not\subseteq$, `!equiv` is $\not\equiv$. (`!=` stays "not equal", $\neq$.)

Set relations — `in` $\in$, `subset` $\subset$, `supset` $\supset$, `subseteq` $\subseteq$, `supseteq` $\supseteq$, `cup` $\cup$, `cap` $\cap$, `setminus` $\setminus$, `emptyset` $\varnothing$ (negate any of them with `!`).

Arrows — `->` or `to` $\to$, `mapsto` $\mapsto$.

```
$forall epsilon > 0, exists eta > 0$
$x in RR setminus QQ$
$x !in QQ$
$(P and Q) => P$
```

Brackets and accents

The integer part and a few one-argument wrappers follow the same `name(...)` shape as `abs` and `vec`: `floor(x)` $\lfloor x \rfloor$, `ceil(x)` $\lceil x \rceil$, `set(x :  x > 0)` a brace set, `abr(u, v)` $\langle u, v \rangle$ an inner product.

An overline is named by intent, the same rule under two words: `bar(...)` for the generic bar (complex conjugate, mean, closure of a set) and `not(...)` for logical negation — so `bar(z)` is $\overline{z}$ and `not(A cup B)` is $\overline{A \cup B}$. The usual accents complete the set: `hat(f)` $\widehat{f}$, `tilde(x)` $\widetilde{x}$.

Arithmetic and sets

A few arithmetic helpers and the set-builder notations:

You write	You get
`lcm(a, b) sign(x) card(A)`	lcm, sgn, card
`powerset(A)`	$\mathcal{P}(A)$
`range(1, n)`	$[\![1, n]\!]$ (integer interval)
`euler(n) mobius(n)`	$\varphi(n)$, $\mu(n)$
`a equiv b mod n`	$a \equiv b \pmod{n}$

2 Algebra

Matrices and systems as blocks, then the named vocabulary of linear maps.

Matrix blocks

Matrices and systems are blocks: **one line is one row**, and inside a matrix `;` separates the entries.

```
<matrix>{
1 ; 2 ; 3
4 ; 5 ; 6
}
```

`<matrix>` draws parentheses, `<det>` the vertical bars of a determinant, `<bmatrix>` square brackets. A single `|` inside a row draws the separation bar of an **augmented matrix**, at the column where you type it; it is allowed on `matrix` and `bmatrix`, never on `det`:

```
<bmatrix>{
2 ; 1 | 7
1 ; -1 | 1
}
```

A `<system>` stacks equations under a brace and aligns them on the first relational operator:

```
<system>{
2x + 3y = 7
x - y = 1
}
```

Linear-algebra vocabulary

Named operators on top of the matrix blocks:

You write	You get
`ker(f) im(f) rank(A)`	$\mathrm{Ker}(f)$, $\mathrm{Im}(f)$, $\mathrm{rg}(A)$
`span(u, v) tr(A) com(A)`	$\mathrm{Span}(u, v)$, $\mathrm{tr}(A)$, $\mathrm{com}(A)$
`transpose(A) inv(A)`	$A^{\top}$ A^{-1}
`eigen(A) adj(A)`	$\mathrm{Sp}(A)$, $\mathrm{adj}(A)$

Counting

The binomial coefficient and arrangement take **two** arguments, so a one-argument `C(t)` stays an ordinary function:

You write	You get
`C(n, k)` `A(n, k)`	$\binom{n}{k}$ A_n^k
`factorial(n)`, `n!`	$n!$

3 Analysis

Limits, operators, functions, derivatives and integrals, the vector operators and transforms, and the full function studies.

Limits with an arrow

`lim(x->0) f(x)` already sets a limit with its target under the word. For the running phrase "u_n tends to ℓ" written over an arrow, use `arrow(...)`: `u_n arrow(n to +inf) l` sets a long right arrow with the condition underneath. `to` and `->` are interchangeable inside it.

```
$lim(n -> +inf) u_n = l$
$u_n arrow(n to +inf) l$
```

Scripts bind tighter than division, as in ordinary reading: `x^2/y` is the fraction $\frac{x^2}{y}$ and `1/i^2` is $\frac{1}{i^2}$, not $\frac{1}{i}^2$. To divide by a base before its script, parenthesise.

Operators with an index

`sum`, `prod` and `lim` carry their index in `(...)` and set their whole expression in display style, so a fraction in the body keeps full size and a limit's target sits *under* the word, as on a blackboard. A `sum`/`prod`/`lim` body runs to the end of the formula or to the first `=`, so the right-hand side of an identity stays outside the operator:

```
$sum(i=1, n) i = n(n+1)/2$
$lim(x->0) sin(x)/x = 1$
$lim(x->+inf) 1/x$
```

Functions and trigonometry

Function names are set upright automatically — no backslashes: `sin cos tan cot sec csc`, the inverses `arcsin arccos arctan`, the hyperbolics `sinh cosh tanh coth`, and `ln log exp det dim gcd deg ker arg max min sup`. A name glued to `(...)` takes its argument as one atom, so fractions and powers behave:

```
$sin(x)^2 + cos(x)^2 = 1$
$tan(x) = sin(x)/cos(x)$
```

Derivatives

A Leibniz derivative is written as the fraction it is, and the differential `d` is set upright (ISO 80000-2), matching the `d` of the integrals. The romanisation triggers *only* when both sides of the fraction carry the differential, so a variable named `d` is never disturbed: `d/2` stays the fraction d over 2.

```
$dy/dx$                 % upright d
$(d^2 y)/(dx^2)$        % second derivative
$dy/dx + y = 0$         % differential equation
```

Partial derivatives use `partial` (∂); parenthesise each side so the fraction groups correctly:

```
$(partial f)/(partial x)$
$(partial u)/(partial t) = (partial^2 u)/(partial x^2)$
```

Inject a computed value with `#{expr}` (or `#name`), including inside maths: `$#k^2$`. Decimal numbers follow the `lang` option.

Differential operators and transforms

The vector operators, the Landau notations (spelt out so `o`/`O` stay free), and the integral transforms:

You write	You get
`grad(f) div(F) curl(F)`	$\operatorname{grad} f$, $\operatorname{div} F$, $\operatorname{rot} F$
`lap(f)`	Δf (no parentheses)
`dirderiv(f, u)`	$\nabla_u f$
`bigO(...) litO(...)`	$O(\cdot)$, $o(\cdot)$
`laplace(f) fourier(f)`	$\mathcal{L}\{f\}$, $\mathcal{F}\{f\}$
`surfint(S) volint(V)`	$\oiint_S$, $\iiint_V$

Integrals

The integral family names its variable and bounds in the head `(...)` and captures the integrand as its body; the differential is appended for you. No bounds gives a primitive, a comma-separated pair a definite integral:

```
$int(x) f(x)$               % primitive
$int(x=a, b) f(x)$          % definite
```

Separate several domains with `;` for multiple integrals: the count of domains chooses $\int$, $\iint$ or $\iiint$, and the differentials come out in reverse order (the Fubini convention). A single named domain is a region integral:

```
$int(x=a, b ; y=c, d) f(x,y)$
$int(D) f$
```

Three named integrals complete the family: `contourint(C) f(z)` a contour integral $\oint$, `pvint(x=a, b) f(x)` a Cauchy principal value, and `meanint(x=a, b) f(x)` the average (normalised) integral.

Function studies

Three tags turn a function into a variation table and a curve. They share one source of truth: a *function object* built once with `<fn>`, then read by `<vartab>` (the table) and `<plot>` (the graph).

The function object.

The equation form registers itself under its left-hand name and reads as the board does; the variable is declared in situ, table attributes stay attributes after the expression:

```
<fn f(x) = (x+2)(x-3)>
<fn g(t) = t^2 - 2t  x:{-inf | 1 | +inf}  deriv:{- | +}
                     var:{+inf \ -1 / +inf}>
```

A tabular-only object takes the head without `=` — `<fn v(x) x:{...}  var:{...}>`. Objects rebind sequentially: a later `<fn f(x) = ...>` shadows the name for what follows, never for what precedes. The 2.4 `name:`/`expr:` attributes still parse but are deprecated; the `let` binding remains available:

```
let k = <fn name:{k(x)}
            expr:{(x^2+1)/(x-1)}
            x:{-inf | 1-sqrt(2) | 1 | 1+sqrt(2) | +inf}
            second:{- | - || + | +}
            deriv:{+ | - || - | +}
            var:{-inf / 2-2sqrt(2) \ -inf || +inf \ 2+2sqrt(2) / +inf}>
```

`let NAME = <fn ...>` stores the object under `NAME` (the assignment may span several lines, up to the closing `>`). The fields are `name:` (function and variable, fixing the row labels), `expr:` (the formula, read only by `<plot>`), `x:` (remarkable abscissas separated by `|`), `deriv:` and `second:` (sign of the first and second derivative, one per interval), and `var:` (the variation line).

The table.

`<vartab k>` reads the object; or write the data inline with `<vartab x:{...}  deriv:{...} var:{...}>`. Only `x:` and `var:` are required, so the table comes in four shapes:

- `second:` + `deriv:` — full study: x, $f''(x)$, $f'(x)$, $f(x)$
- `deriv:` — classic variation table: x, $f'(x)$, $f(x)$
- `second:` — convexity table: x, $f''(x)$, $f(x)$
- neither — plain value table: x, $f(x)$

Sign lines take one sign per interval (`+`, `-`, or empty); `||` marks a forbidden value and crosses every row; zeros at sign changes are placed automatically. In `var:`, a connector is `/` (rising) or

\ (falling) *surrounded by spaces* — glued, as in `2/3`, it is a fraction. A value between two arrows of the same direction is rejected: the table lists only bounds and extrema.

The sign table.

`<signtab>` is `<vartab>`'s companion for factored expressions. The automatic form *computes* the rows from a product or quotient of affine factors — rational zeros printed exactly, a forbidden value at each denominator zero, even multiplicities known not to change sign:

```
<fn f(x) = (x+2)(x-3)>
<signtab f>

<fn g(x) = (x+1)/(x-2)>
<signtab g>
```

The manual block form is the general one, for rows the affine engine cannot derive (a trigonometric factor, an irreducible quadratic): one row per factor, usually a conclusion row last.

```
<signtab x:{-inf | -2 | 3 | +inf}>{
  x+2  : - | 0 | + | +
  x-3  : - | - | 0 | +
  f(x) : + | 0 | - | 0 | +
}
```

A cell is a sign per interval; a `0` marks a zero at the boundary it follows, `||` a forbidden value. The invariant is checked: a row that changes sign without a `0` or a `||` at the boundary is a compilation error, not a wrong table in print — an expression cannot change sign without vanishing or ceasing to exist.

The graph.

```
<plot k samples:200 x:{-4, 6} y:{-10, 12}>
```

`<plot k>` reads `expr:` and `name:` from the object. `x:{a, b}` is the horizontal display window (defaults to the table's finite abscissas), `y:{c, d}` the vertical window (needed when the function runs to infinity), `samples:N` the number of points. The formula is translated to the plotter's syntax (variable renamed, trigonometry in degrees, implicit products made explicit) and poles are handled so no spurious line joins the branches. Supported in `expr:`: `+ - * / ^`, parentheses, `sin cos tan exp ln log sqrt abs`, and `pi`.

`<vartab>` is declarative, `<plot>` is computed: keeping `var:` consistent with `expr:` is the author's responsibility.

Parametric and polar curves.

A curve that is not a function graph is the same `<plot>` with `kind:`. `kind:parametric` reads `expr:{x(t), y(t)}` — two components in the parameter `t` — over the window `t:{a, b}`. `kind:polar` reads `expr:{r(theta)}` over `theta:{a, b}` and is drawn as the parametric pair $(r\cos\theta, r\sin\theta)$. Bounds may use `pi` (`t:{0, 2pi}`), and `x: / y:` set the display window as before.

```
let ell = <fn expr:{3cos(t), 2sin(t)}>
<plot ell kind:parametric t:{0, 2pi} x:{-4, 4} y:{-3, 3}>

let cardio = <fn expr:{1+cos(theta)}>
<plot cardio kind:polar theta:{0, 2pi}>
```

A conic is drawn through its parametrisation: an ellipse as `{a cos(t), b sin(t)}`, a parabola as `{t, t^2}`, a branch of a hyperbola as `{a cosh(t), b sinh(t)}` — no separate conic directive, the parametric form already names them all.

Areas.

`area:{a, b}` shades the region between the curve and the x-axis on $[a, b]$, dashed verticals at the bounds; `between:g` shades between two curves instead, clipped to `area:` when given:

```
<plot f x:{-1, 4} area:{1, 3}>
<plot f x:{-1, 4} between:g area:{0, 3}>
```

Sequences: the cobweb.

`cobweb:{u0}` (or `cobweb:{u0, n}`, default $n = 8$) draws the bisector $y = x$ and the staircase of $u_{n+1} = f(u_n)$, the iterates computed at compile time and the first three labelled on the axis:

```
let h = <fn name:{h(x)} expr:{sqrt(2x+3)}>
<plot h x:{0, 4} y:{0, 4} cobweb:{0.2, 10}>
```

Probability laws.

The two laws of the lycée need no `<fn>` object; `area:` highlights $P(a \leq X \leq b)$ on both — shaded under the Gaussian, stronger bars on the binomial:

```
<plot normal:{0, 1} area:{-1, 1.5}>
<plot binomial:{12, 0.4} area:{3, 6}>
```

The normal window defaults to $\mu \pm 4\sigma$; the binomial masses are computed exactly.

The number line.

`<numberline>` draws the graduated line with interval sets — the figure of inequalities and absolute values:

```
<numberline x:{-5, 5} set:{[-2, 3) union (4, inf)} points:{-4}>
```

The block form *solves*: it takes one inequality (a tag's braces cannot carry `<` and `>`) and computes the set exactly — an affine expression against a constant, a product or quotient of affine factors against 0, or `abs(affine)` against a constant. Strictness decides open or closed; a denominator zero is always excluded; an isolated solution is a point.

```
<numberline x:{-5, 5}>{
  abs(x - 1/2) >= 5/2
}
```

Brackets follow the international convention (`[ ]` included, `( )` excluded); a bound may be `inf`, the segment then runs to the arrow. `points:` pins isolated values. One unit is one centimetre, halved automatically while the line exceeds the page.

4 Probability

Counting feeds probability; the blackboard operators are the doubled letters `PP` and `EE`, with the conditional bar written `mid`.

You write	You get
`PP(A)`, `PP(A mid B)`	$\mathbb{P}(A)$, $\mathbb{P}(A \mid B)$
`EE(X)`	$\mathbb{E}(X)$
`var(X) std(X) cov(X, Y)`	$\mathrm{Var}(X)$, $\sigma(X)$, $\mathrm{Cov}(X, Y)$
`normal(mu, sigma)`	$\mathcal{N}(\mu, \sigma^2)$
`poisson(lambda) binomial(n, p)`	$\mathcal{P}(\lambda)$, $\mathcal{B}(n, p)$
`repart(X, x) densite(X, x)`	$F_X(x)$, $f_X(x)$

The weighted tree.

`<tree>` is the most-drawn object of première and terminale. Each line reads `LABEL PROBABILITY`, an optional brace opening the children; a leading `!` is the complementary event, rendered with the overline. Probabilities may be numbers or symbols (`p`, `1-p`), both typeset through the math mini-language.

A node with exactly one branch receives its complement automatically — the only case that is mathematically determined; sibling probabilities that do not sum to 1 are a compilation error.

```
<tree products:on>{
  A 0.3 {
    B 0.6
  }
  !A 0.7 {
    B 0.1
  }
}
```

`products:on` writes $P(A \cap B) = 0.18$ at the end of each leaf: the exact product when every probability on the path is numeric, the symbolic chain otherwise. The layout is computed — leaves evenly spaced, each node at the mean height of its children. No loops inside the block: a tree is written by hand, that is the point of a tree.

5 Statistics

Descriptive statistics is one tag, `<stats>`, five `kind:`s, every number computed at compile time:

```
<stats kind:bars data:{1: 4 | 2: 7 | 3: 2}>
<stats kind:bars data:{Walk: 5 | Bus: 3 | Bike: 2}>
<stats kind:histogram bounds:{0, 5, 10, 20} counts:{3, 7, 2}>
<stats kind:pie data:{Walk: 5 | Bus: 3 | Bike: 2}>
<stats kind:boxplot data:{12, 15, 9, 21, 14, 15, 18}>
<stats kind:scatter data:{(1, 2.1) (2, 2.6) (3, 3.4)} fit:on>
```

Three honesty guarantees, because a statistics figure teaches by its construction: the histogram draws *density* (count over class width), so unequal classes stay truthful; the boxplot uses the French secondary-school quartiles (Q_1 the smallest value with at least a quarter of the data at or below it, Q_3 likewise at three quarters); `fit:on` draws the least-squares line, computed, not eyeballed. The pie starts at twelve o'clock and runs clockwise, percentages appended to the labels. The dictionary `key: value | ...` is the form of the mappings (bars — numeric keys give a numeric axis, words give categories — and pie); pairs `(x, y)` remain the form of the clouds, where an abscissa may repeat.

A Lua table declared with `let` is passed by name — the home of data reused across figures:

```
let notes = {
    Mathematiques = 15.5,
    Francais = 12.0,
    Anglais = 18.0
}
<stats kind:bars data:notes>
<stats kind:pie data:notes>
```

A Lua table has no writing order: map keys come out *sorted* (numbers numerically, words alphabetically); when the order matters, a list of pairs keeps it: `let t = {{"T1", 11.5}, {"T2", 13}}`. Lists of numbers feed `boxplot`, lists of pairs feed `scatter`, two list variables feed `histogram` through `bounds:` and `counts:`.

Part V Programming the document

1 Block aliases

Define a reusable component once; `#param` placeholders are filled at the call site, and the call-site body becomes the block content.

```
let card{title, frame} = <box title:{#title} line:#frame radius:2>

<card First, Crimson>{
Called with two arguments. No box options to repeat.
}
```

```
<card Second, Navy>{
Same component, different look.
}
```

2 Control flow

```
for n in 1..3 {            % numeric range
<c Navy b>Sheet #n
}

for f in [chat.png, chien.png] {   % explicit list
<img 16>{#f}
}

if score >= 10 {
<Green>Passed.
} else {
<Red>Try again.
}
```

Loops and conditions work in the document body, inside boxes, and inside table bodies. The loop variable interpolates everywhere via `#`.

Part VI Reference

1 Security

scholatex evaluates `let name = expr`, `#{expr}` and the conditions of `for`/`if`/`while` as Lua at compile time, so by default a document can run arbitrary code — exactly like `\directlua`.

Setting `untrusted=true` runs that Lua in a restricted environment: only pure, side-effect-free names are visible (the maths the document needs plus `string`/`table` helpers), while `os`, `io`, `package`, `require`, `load`, `debug`, `setmetatable` and `getmetatable` are absent. A blocked access stops the compile with a clear message; a runaway loop is aborted by an instruction-count ceiling. Because the step counter cannot interrupt work inside a single C call, `string.rep` and `string.format` are capped — both through the global `string` table and through method calls on a string value (`("x"):rep(n)`), which a restricted string metatable routes to the same capped functions.

Scope. `untrusted` hardens the scholatex expression layer only. It does *not* sandbox LuaLaTeX as a whole: a hostile `.tex` can still call `\directlua`, `\write18` (with shell-escape), `\input`, and so on. To compile a whole `.tex` you do not trust, run `lualatex` **without** `-shell-escape`, ideally inside a container.

2 Examples

The `examples/` folder contains seven self-contained, fully commented documents that together exercise every feature:

File	Covers
`text-style.tex`	case rule, styles, colours, fonts, sizes, alignment, escapes, aliases, TOC
`containers.tex`	tables, boxes and the named-area grid
`basics.tex`	mini-language, sets, logic, negation, integer part, the overline, arithmetic hel
`algebra.tex`	matrix, determinant, augmented matrix, systems, and the linear-algebra voc
`analysis.tex`	operators, limits, trigonometry, derivatives, the vector operators, integrals, t
`statistics-probability.tex`	counting, probability, variance, distributions — then trees, the two laws and

Compile any of them with `lualatex <file>.tex` from the `examples/` folder. The image folders `IMG/` and `IMAGES/PNG/` ship alongside them; `containers.tex` uses `imgdir={IMG, IMAGES/PNG}` to show that bare image names are resolved across several directories.

3 Diagnostics

Errors point at the source line, e.g. `scholatex: line 12: unknown tag attribute: 'xyz'`. Defining an alias whose name is a built-in (`let section = ...`, `let tab = ...`) prints a warning: the built-in always wins, so the alias would be silently dead — pick a different name.