

blockst Version 0.4.2

Render Scratch, Blockly, MakeCode and Open Roberta blocks for educational documents.

blockst renders programming blocks directly in Typst documents — Scratch, Blockly with a jwinf profile, MakeCode for the micro:bit and the Calliope mini, and Open Roberta NEPO — for worksheets, tutorials and teaching material. Block code is written as plain text and rendered by a bundled WASM plugin, Scratch in 26 languages including right-to-left scripts, with a turtle-graphics execution engine and helpers for importing real `.sb3` project files.

AUTOREN Loewe1000
LIZENZ MIT
VERWEISE GitHub

Inhalt

1 Introduction **4**
1.1 About blockst 4
1.2 Quick start 4
1.3 Which function for which editor 5
1.4 Package information 6
2 The notation **7**
3 Options for every editor **8**
3.1 set-blockst() — Global Defaults 8
3.2 blockst() — Group Override Container 8
3.3 Themes 8
3.4 Colours 9
3.5 Fonts 10
3.6 Scale and inset scale 10
3.7 Line numbers 10
4 Scratch **11**
4.1 scratch() — Render Scratch Blocks 11
4.2 raw-scratch — Code fences 11
4.3 Languages 12
4.3.1 Right-to-left languages 13
4.4 `@category` — Quick Color Defaults 14
4.5 scratch-parse() — Parse to AST 14
5 Blockly and jwinf **15**
5.1 blockly() — Render Blockly Blocks 15
5.2 Profiles 15
5.3 Writing jwinf tasks 16
5.4 Languages 16
5.5 Code fences and parsing 16
6 MakeCode **17**
6.1 makecode() — Render MakeCode Blocks 17
6.2 Profiles 18
6.3 Writing MakeCode programs 18
6.4 Languages 18
6.5 Code fences and parsing 18
7 NEPO (Open Roberta) **19**
8 Labels and line numbers **20**
8.1 blockst-labels() — Query Labels 20
8.2 scratch-labels() — Extract Labels 20
8.3 blockst-register-labels() — Pre-register Labels 20
9 Running Scratch programs **22**
9.1 scratch-run Module 22
9.1.1 scratch-run.stage() — Stage Canvas 22
9.1.2 scratch-run.grid() — Coordinate Grid 22
9.1.3 set-scratch-run() — Run Global Defaults 22

9.1.4 Supported Execution Commands ... 23
9.1.5 Complete Example ... 24

10 Importing Scratch projects (SB3) ... 27
10.1 Basic Workflow ... 27
10.2 render-sb3-scripts() ... 27
10.3 render-sb3-variables() ... 28
10.4 render-sb3-lists() ... 28
10.5 Standalone Monitors ... 29
10.5.1 list-monitor() ... 29
10.5.2 variable-monitor() ... 29
10.6 screen-preview() ... 29
10.7 Image Helpers ... 29
10.8 Catalog Helpers ... 30

11 Scratch block catalog ... 31
11.1 Motion ... 31
11.2 Looks ... 32
11.3 Sound ... 34
11.4 Pen ... 35
11.5 Variables ... 36
11.6 Lists ... 36
11.7 Events ... 37
11.8 Control ... 38
11.9 Sensing ... 40
11.10 Operators ... 41

12 Blockly block catalog ... 43
12.1 Standard blocks ... 43
12.1.1 Schleifen ... 43
12.1.2 Logik ... 43
12.1.3 Mathe ... 44
12.1.4 Text ... 45
12.1.5 Listen ... 45
12.1.6 Variablen ... 46
12.1.7 Funktionen ... 46
12.2 jwinf ... 46
12.2.1 Start ... 46
12.2.2 Aktionen ... 46
12.2.3 Schildkroete ... 48
12.2.4 TurtleInput ... 49
12.2.5 Schleifen ... 49
12.2.6 Text ... 49
12.2.7 Listen ... 49
12.2.8 Debug ... 50

13 MakeCode block catalog ... 51
13.1 micro:bit ... 51
13.1.1 Basic ... 51
13.1.2 Input ... 52
13.1.3 Music ... 54

13.1.4 Led ... 56
13.1.5 Radio ... 57
13.1.6 Loops ... 58
13.1.7 Logic ... 59
13.1.8 Variables ... 60
13.1.9 Math ... 60
13.1.10 Functions ... 61
13.1.11 Arrays ... 62
13.1.12 Text ... 63
13.1.13 Game ... 64
13.1.14 Images ... 66
13.1.15 Pins ... 66
13.1.16 Serial ... 69
13.1.17 Control ... 70
13.2 Calliope mini ... 71
13.2.1 Basic ... 71
13.2.2 Input ... 72
13.2.3 Music ... 73
13.2.4 Led ... 73
13.2.5 Functions ... 73
13.2.6 Arrays ... 73
13.2.7 Text ... 74
13.2.8 Game ... 74
13.2.9 Images ... 74
13.2.10 Pins ... 74
13.2.11 Serial ... 75
13.2.12 Motors ... 76
14 Contributing ... 77
15 API Reference ... 78
15.1 `blockly` ... 78
15.2 `blockly-parse` ... 79
15.3 `blockst` ... 79
15.4 `blockst-labels` ... 80
15.5 `blockst-register-labels` ... 80
15.6 `makecode` ... 80
15.7 `makecode-parse` ... 81
15.8 `raw-blockly` ... 81
15.9 `raw-makecode` ... 81
15.10 `raw-scratch` ... 81
15.11 `scratch` ... 82
15.12 `scratch-execute` ... 82
15.13 `scratch-labels` ... 82
15.14 `scratch-parse` ... 83
15.15 `set-blockst` ... 83

1 Introduction

1.1 About blockst

blockst renders block-based programs in Typst documents, each drawn the way its editor draws it:

- **Scratch 3** — `scratch()`, the Scratch look in 26 languages, including right-to-left scripts, with an execution engine for turtle graphics and helpers for importing `.sb3` project files.
- **Blockly** — `blockly()`, today's flat Blockly and the pre-2019 look, with profiles for the Jugendwettbewerb Informatik (jwinf.de) and its robot and turtle blocks.
- **MakeCode** — `makecode()`, the micro:bit and Calliope mini editors' blocks in all 36 of their languages.
- **NEPO (Open Roberta)** — `nepo()`, the Calliope mini and micro:bit blocks of Open Roberta Lab.

All four share one text notation: a block per line, `(…)` for a value, `[… v]` for a dropdown, `<…>` for a condition, indentation and an end marker for the body of a loop. Typst hands the text to a bundled WASM plugin, the plugin parses it and returns SVG, Typst embeds the SVG. Nothing has to be installed beyond the package.

CORE DESIGN

- Fully text-based: write blocks as plain text, get rendered blocks. The same notation for every editor.
- Each editor's own geometry, palette and vocabulary — jwinf's classic Blockly measured on jwinf.de, MakeCode's zelos renderer as the editors run it.
- Localized: Scratch's official translations, Blockly's message files, the MakeCode editors' own strings.
- Themes for print (`print`, `grayscale`, `high-contrast`) and the document's own category colours over any palette.
- Line numbers and line labels for worksheets that talk about single lines.

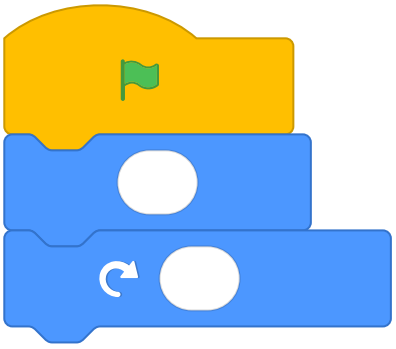
BREAKING CHANGE SINCE 0.2.0

The old pre-0.2.0 native-Typst renderer syntax is **removed**. From 0.2.0 onward, blockst uses only the text-to-WASM pipeline. Documents that still rely on the previous syntax must be migrated.

1.2 Quick start

```
#import "@preview/blockst:0.4.2": scratch, blockly, makecode, nepo

#scratch("
when green flag clicked
move (10) steps
turn cw (15) degrees
")
```



The same document can hold the other editors' blocks. A jwinf task:

```
#blockly("
Roboter-Programm
wiederhole (4) mal:
  gehe nach rechts
  falls <auf Kiste>
    hebe Murmel auf ::aktionen
  ende
ende
", profile: "jwinf")
```

A micro:bit program:

```
#makecode("
beim Start
  zeige Symbol [Herz v]
ende
wenn Knopf [A v] geklickt
  zeige Zahl ((1) + (2))
  pausiere (ms) (100)
ende
")
```

1.3 Which function for which editor

Editor	Function	Profiles	Default language	Code fences
Scratch 3	`scratch()`	—	`"en"`, 26 languages	`scratch` via `raw-scratch()`
Blockly, jwinf	`blockly()`	`blockly` (default), `blockly-klassisch`, `jwinf`, `jwinf-turtle`	`"de"`, 24 languages	`blockly`, `jwinf`, `jwinf-turtle` via `raw-blockly()`
MakeCode	`makecode()`	`makecode` (micro:bit, default), `makecode-calliope`	`"de"`, 36 languages	`makecode`, `microbit`, `calliope` via `raw-makecode()`
Open Roberta	`nepo()`	platforms `calliope` (default), `calliopev3`, `microbit`	`"de"`	`nepo` via `raw-nepo()`

Every function takes the same rendering options — theme, scale, font, line numbers, colours — and all of them read the defaults set with `set-blockst()`. The chapters Scratch, Blockly and jwinf, MakeCode and NEPO describe what is particular to each editor.

1.4 Package information

- **Version:** 0.4.2
- **License:** MIT
- **Repository:** github.com/Loewe1000/blockst
- **Compiler requirement:** Typst 0.15.0+
- **Font requirement:** Scratch and Blockly labels are designed for Helvetica Neue. On Linux/Windows install a compatible font (e.g. Nimbus Sans) or set another one with `set-blockst(font: …)`. MakeCode labels use a monospace face — Menlo, Consolas or DejaVu Sans Mono, whichever is installed.

2 The notation

Blocks are written one per line, the way they read in the editor. Inputs are marked by their brackets, a body is indented and closed by an end marker. The vocabulary is the editor's own in the chosen language — the parser matches against Scratch's translations, Blockly's message files or the MakeCode editors' strings, and a line it does not recognise is drawn as written in a neutral grey (or in the colour of a category you name, see below).

Pattern	Example	Description
Stack block	`move (10) steps`	A command; blocks on consecutive lines snap together
Hat block	`when green flag clicked`	An event; starts a new script
Reporter	`(x position)`	A value with round ends
Boolean	`<mouse down?>`	A condition with pointed ends
Input: number	`(10)`	Numeric input field
Input: string	`[hello]`	String input field
Input: dropdown	`[random position v]`	Dropdown selector — the `v` marks it
Empty input	`()`, `[]`, `<>`	An empty slot
C-block	`repeat (4)` … `end`	A block with a body: indent the body, close it with the end marker
C-block with else	`if <…> then` … `else` … `end`	A second body after the else marker
Icon	`@greenFlag`, `@turnRight`, `@turnLeft`	The flag and the turn arrows
Line label	`move (10) steps #step`	Names the line, see Labels and line numbers

What differs between the editors is small and follows the editor:

- **End and else markers** follow the language: `end` / `else` in English, `ende` / `sonst` in German Blockly, `ende` / `ansonsten` in German MakeCode, `fin` / `sinon` in French. `end`, `ende` and `else` are understood in every Blockly and MakeCode language.
- **Booleans.** Scratch and MakeCode draw a condition as a hexagon, so `<…>` and `(…)` are different shapes. Blockly has no hexagonal boolean: there `<…>` equals `(…)`.
- **Unknown labels.** Scratch names a category with a prefix, `@motion free text` (see @category). Blockly and MakeCode use a suffix, `hebe Murmel auf ::aktionen` — on jwinf the normal case, because the same block reads differently from task to task.
- **Units.** MakeCode shows some units in parentheses; they are typed like a value and drawn as the label: `pausiere (ms) (100)`, `Temperatur (°C)`.
- **Editors.** MakeCode's LED matrix and melody editor have a notation of their own, see MakeCode.

3 Options for every editor

`scratch()`, `blockly()`, `makecode()` and `nepo()` accept the same rendering options. Given on the call, they apply to that block group; set with `set-blockst()`, they become the document's defaults; wrapped in `blockst[…]`, they apply to everything inside.

3.1 set-blockst() — Global Defaults

```
#set-blockst(
  theme: none,            // "normal", "high-contrast", "print", "grayscale"
  profile: none,          // default profile for blockly() and makecode()
  colors: none,           // (category: colour, …) over any palette
  scale: none,            // 80%, 0.8 — overall size
  inset-scale: none,      // 60% thin, 125% thick blocks; text size unchanged
  stroke-width: none,
  font: none,             // "Nimbus Sans"
  language: none,         // default language for every editor
  line-numbers: none,
  line-number-start: none,
  line-number-first-block: none,
  line-number-gutter: none,
)
```

All parameters are optional. Only provided values override the current defaults; a later `set-blockst()` changes only what it names.

3.2 blockst() — Group Override Container

```
#blockst(theme: "high-contrast", scale: 80%)[
  #scratch("when green flag clicked\nmove (10) steps")
  #blockly("wiederhole (4) mal:\n  gehe nach rechts\nende", profile: "jwinf")
]
```

All parameters match `set-blockst()` but apply only within the body; `spacing` (default `1.5em`) sets the gap between the blocks inside.

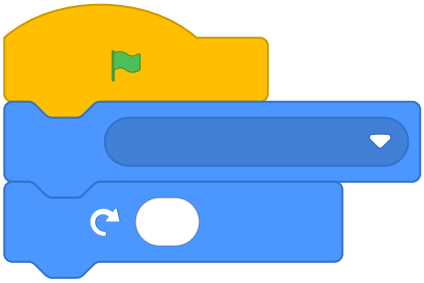
3.3 Themes

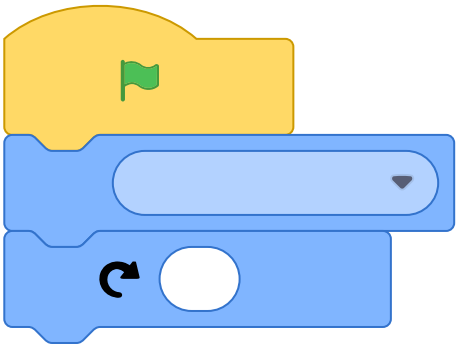
Theme	Description
`"normal"`	Default — the editor's full colours.
`"high-contrast"`	Lighter, high-contrast variant for accessibility.
`"print"`	Every block white with a black outline. The lightest on ink, but all categories look alike.
`"grayscale"`	One distinct grey per category, so a control block still reads differently from an operator on a monochrome page.

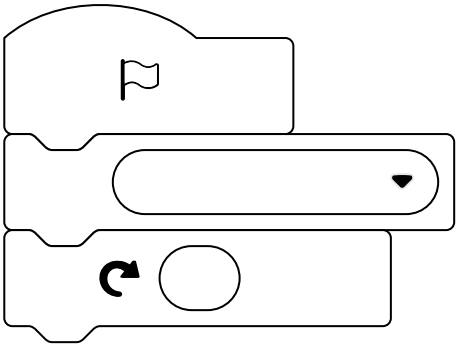
The derived themes are computed from the palette in use, so `grayscale` on a jwinf profile is jwinf's categories in grey, and a colour set with `colors` carries its own bevel, stroke, high-contrast and grey shades.

```
#let script = "when green flag clicked
go to (random position v)
turn cw (30) degrees"

#blockst(inset-scale: 50%)[#scratch(script)]
#blockst(theme: "high-contrast")[#scratch(script)]
#blockst(theme: "print")[#scratch(script)]
```







3.4 Colours

`colors` lays the document's own category colours over any palette — for the document with `set-blockst()`, for a single block or group on the rendering function or on `blockst()`. A hex string or a Typst colour per category; the category names are the ones the editor uses (`motion`, `looks`, … for Scratch, `aktionen`, `schleifen`, `logik`, … for jwinf, `basic`, `input`, `loops`, … for MakeCode).

```
#set-blockst(colors: (logik: "#73cc47"))
#blockly(programm, profile: "jwinf", colors: (aktionen: "#cc7347", schleifen:
rgb("#5ba55b")))
```

jwinf colours its categories per task family; `colors` is the way to match a task that deviates from both jwinf profiles.

3.5 Fonts

```
#set-blockst(font: "Nimbus Sans")
```

Scratch and Blockly labels default to Helvetica Neue, MakeCode labels to Menlo, Consolas or DejaVu Sans Mono. Set the font here rather than with `set text(font: …)`: the renderer measures each label and writes the font into the SVG, so a document-level `set text` changes what is drawn but not what was measured, and the labels overlap.

3.6 Scale and inset scale

`scale` multiplies the whole drawing (`80%`, `0.8`). `inset-scale` changes the thickness of the blocks without touching the text: `60%` gives compact, `125%` generous blocks.

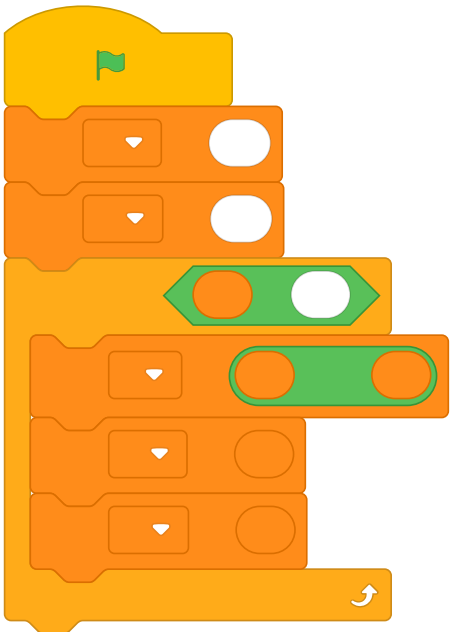
```
#set-blockst(scale: 80%)
#set-blockst(inset-scale: 60%)   // compact blocks
```

3.7 Line numbers

```
#set-blockst(line-numbers: true, line-number-start: 1, line-number-gutter: 24)
```

`line-number-start` is the first number, `line-number-gutter` the width of the gutter in points. Numbering continues from one block group to the next; `line-number-first-block` says with which group it starts. The three can also be given as one dictionary, `line-numbering: (enabled: true, start: 1, first-block: 1)`.

Euclidean Algorithm (gcd label walkthrough)



Explanation of labeled lines

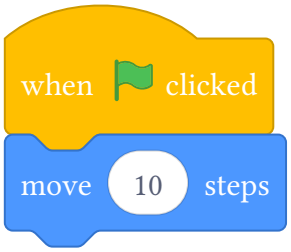
- **Line 4**: Loop condition `b != 0` controls termination.
- **Line 5**: Core rule: `r = a mod b`.
- **Line 7**: State update that advances to the next pair.

4 Scratch

4.1 scratch() — Render Scratch Blocks

Parses Scratch text and renders the blocks the way Scratch 3 draws them.

```
#scratch("when green flag clicked\nmove (10)
steps")
```




```
#scratch(
  text,
  language: "en",
  theme: auto,
  scale: auto,
  font: auto,
  colors: auto,
  line-numbers: auto,
  line-number-start: auto,
  line-number-gutter: auto,
  inset-scale: auto,
)
```

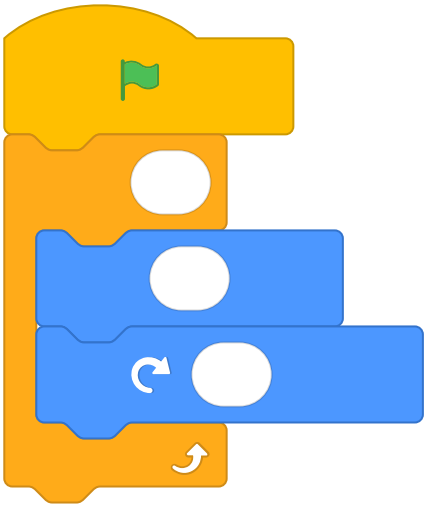
`text` is the Scratch block text in the chosen language; the other parameters are the options every editor shares, `auto` meaning the document's default. The parser knows the full Scratch 3 vocabulary in all 26 locales, including the pen extension.

4.2 raw-scratch — Code fences

The `raw-scratch()` show rule renders `scratch` code fences as blocks:

```
#show: raw-scratch(language: "en")
```

```
when green flag clicked
repeat (4)
  move (30) steps
  turn cw (90) degrees
end
```

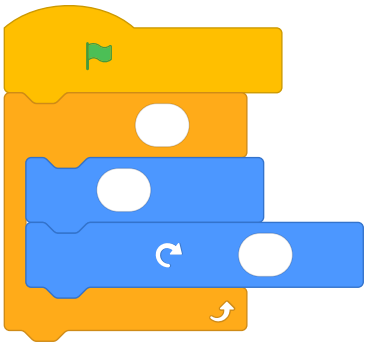


4.3 Languages

Scratch blocks come in **26 languages**, from Scratch's official translations. Set the language on the call or with `set-blockst(language: …)`; block text must match the chosen locale's vocabulary.

Code	Language	Code	Language
`"en"`	English	`"de"`	German
`"fr"`	French	`"es"`	Spanish
`"it"`	Italian	`"nl"`	Dutch
`"pt"`	Portuguese	`"pl"`	Polish
`"ru"`	Russian	`"ja"`	Japanese
`"ca"`	Catalan	`"cs"`	Czech
`"cy"`	Welsh	`"el"`	Greek
`"fa"`	Persian	`"gd"`	Scottish Gaelic
`"he"`	Hebrew	`"hi"`	Hindi
`"hr"`	Croatian	`"hu"`	Hungarian
`"id"`	Indonesian	`"nb"`	Norwegian (Bokmål)
`"ro"`	Romanian	`"sl"`	Slovenian
`"tr"`	Turkish	`"ar"`	Arabic

```
#set-blockst(scale: 67.5%)
#scratch("
Wenn die grüne Flagge angeklickt
wiederhole (4) mal
  gehe (30) er Schritt
  drehe dich nach rechts um (90) Grad
end
", language: "de")
```

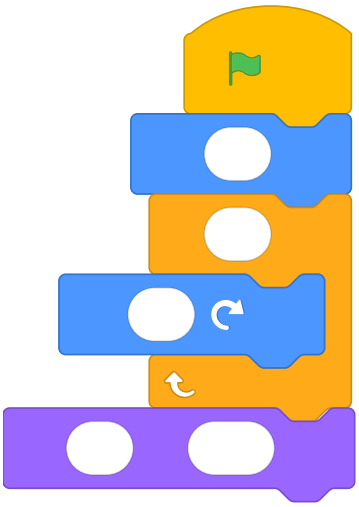


4.3.1 Right-to-left languages

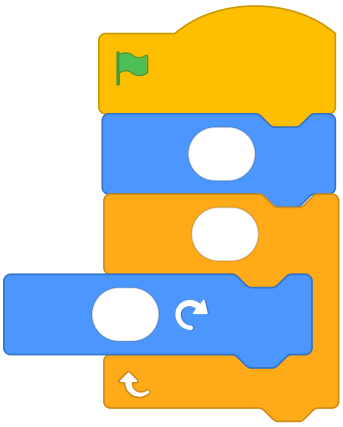
Arabic (`"ar"`), Hebrew (`"he"`) and Persian (`"fa"`) render right-to-left. Nothing has to be switched on: each locale declares its own direction, and the renderer mirrors the layout — the notch, the hat dome, the C-block mouth, the loop arrow, the pen badge, the `define` hat and the order of the labels all move to the reading edge.

```
#scratch("
عند نقر @greenFlag
تحرك خطوة (10)
كرِّر مرة (4)
استدر @turnRight درجة (90)
نهاية
", language: "ar")
```

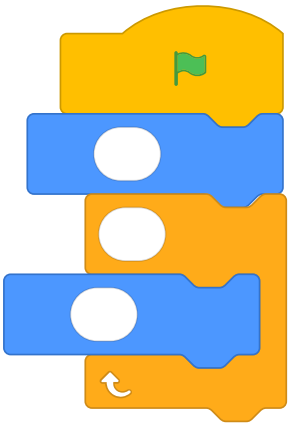
العربية



עברית



فارسی



> **WHAT IS MIRRORED, AND WHAT IS NOT**
> Layout is mirrored; meaning is not. The loop arrow points back to the top of the loop, which is a claim about the drawing, so it flips with the drawing. `@turnRight` and `@turnLeft` are never mirrored — their direction is what the sprite is being told to do, and a mirrored `@turnRight` would tell the reader to turn the other way.

Arabic short vowels are optional and their order is not canonical, so blocks match whether or not you type the harakat: `كرِّر` and `كرر` both find the repeat block.

4.4 `@category` — Quick Color Defaults

A `@category` prefix forces a block's category colour, even without matching the full localized syntax.

```
@motion         // → default motion block
@motion free text  // → unrecognized block in motion color
```

Token	Normalized	Default Block
`@motion`	motion	`move (10) steps`
`@looks`	looks	`say [Hello!]`
`@sound`	sound	`play sound [pop v]`
`@events`	events	`when green flag clicked`
`@control`	control	`repeat (10)`
`@sensing`	sensing	`ask [What's your name?] and wait`
`@operators`	operators	`(() + ())`
`@variable`	variables	`set [var v] to (0)`
`@list`	lists	`add (thing) to [list v]`
`@pen`	pen	`clear`
`@event`	events	(alias for `events`)
`@operator`	operators	(alias for `operators`)

When `@category` is followed by text that matches a known block in that category, the actual block is rendered; otherwise the fallback is an `unrecognized` block in the forced colour.

```
@list add (12) to [my list v]    // → DATA_ADDTOLIST
@variable change [score v] by (1) // → DATA_CHANGEVARIABLEBY
```

4.5 scratch-parse() — Parse to AST

Parses Scratch text to an abstract syntax tree for programmatic use — a nested structure of blocks, inputs and bodies.

```
#scratch-parse(text, language: "en")
```

The complete list of Scratch blocks, rendered live, is the Scratch block catalog at the end of this manual. Running Scratch programs as turtle graphics and importing `.sb3` files have chapters of their own.

5 Blockly and jwinf

5.1 blockly() — Render Blockly Blocks

`blockly()` renders Blockly blocks from the shared notation, drawn with Blockly's shapes — notch, puzzle tab, boxed fields, the mutator gear on an if block, the warning sign on a loop-control block outside a loop.

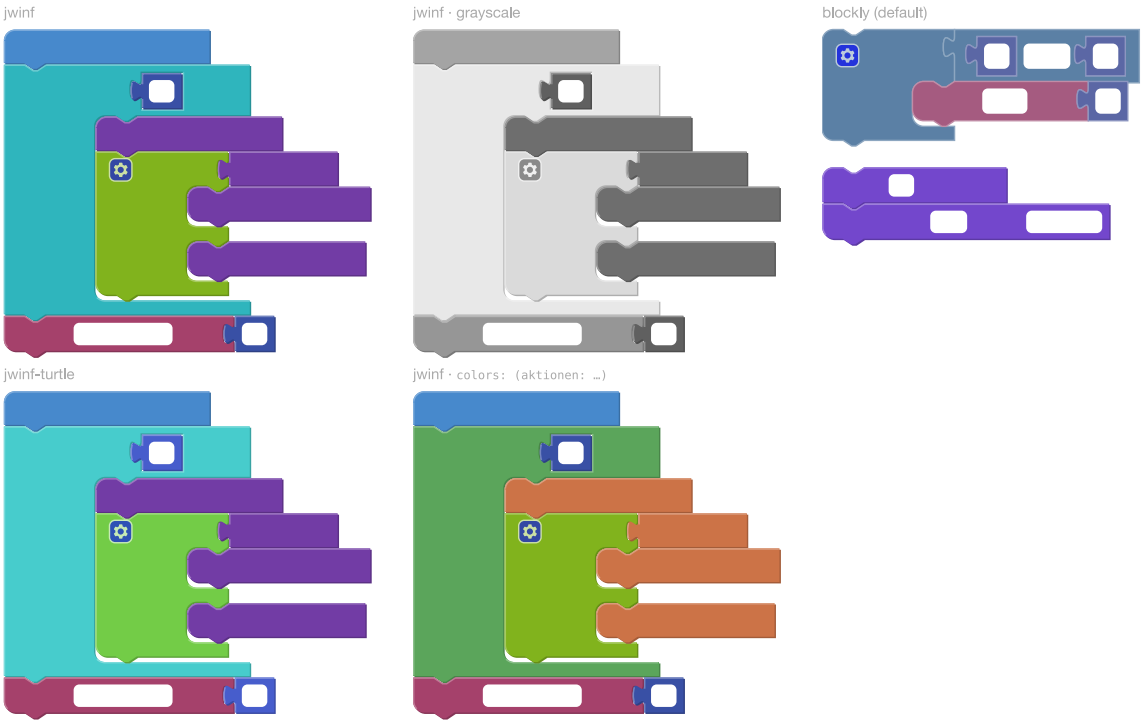
```
#blockly(
  text,
  profile: auto,      // "blockly", "blockly-klassisch", "jwinf", "jwinf-turtle"
  language: auto,     // "de" by default
  theme: auto,
  scale: auto,
  font: auto,
  colors: auto,
  line-numbers: auto,
  line-number-start: auto,
  line-number-gutter: auto,
  inset-scale: auto,
)
```

5.2 Profiles

A profile chooses look and vocabulary. `set-blockst(profile: …)` sets the default for `blockly()`.

Profile	Look	Vocabulary
`"blockly"` (default)	today's flat Blockly (thrasos)	Blockly's standard blocks
`"blockly-klassisch"`	Blockly before 2019, with the bevel highlight	same
`"jwinf"`	jwinf.de — the classic geometry measured on the site, the palette of the robot training tasks	robot and turtle world blocks, plus the standard blocks in the old wording
`"jwinf-turtle"`	the same, with the colours of the Freie Turtle-Umgebung	same

```
#blockly("
Roboter-Programm
wiederhole (4) mal:
  gehe nach rechts
  falls <auf Kiste>
    hebe Murmel auf ::aktionen
  ende
ende
", profile: "jwinf")
```



5.3 Writing jwinf tasks

On jwinf the same block reads differently from task to task — „hebe Murmel auf„, „nimm Fisch“, „Holzstapel einsammeln„ — so there is no fixed vocabulary to match against. Write the label as it appears and name the category with a `::` suffix: `aktionen`, `schildkroete`, `sensoren`, `schleifen`, `logik`, `mathe`, `variablen`, `funktionen`, `text`, `listen`, `ausgeben`, `einlesen`. Blocks the profile knows — loops, conditions, variables, the robot and turtle commands — are recognised without a suffix.

`ende` closes a C-block, `sonst` opens its else branch, and `<…>` equals `(…)`: Blockly has no hexagonal boolean. Where a task's colours deviate from both profiles, `colors` lays the document's own over the palette (see Colours).

5.4 Languages

The standard blocks come in 24 languages, from Blockly's own message files: `language: "de"` (the default), `"en"`, `"fr"` (`répéter (10) fois`), `"ja"` (`(10) □□□□□`), and ar, ca, cs, el, es, fa, he, hi, hr, hu, id, it, nb, nl, pl, pt, ro, ru, sl, tr. The jwinf world blocks exist in German only. Each language closes a C-block with the marker its Scratch locale uses (`ende`, `end`, `fin`, …) and opens the else branch with Blockly's word (`sonst`, `else`, `sinon`, …); `ende`, `end` and `else` work in every language.

5.5 Code fences and parsing

`raw-blockly()` renders `blockly` fences with the default profile, `jwinf` fences with the jwinf profile and `jwinf-turtle` fences with the turtle colours, whatever the arguments say. `blockly-parse()` returns the AST.

```
#show: raw-blockly()
#blockly-parse(text, language: "de", profile: "blockly")
```

Themes apply as for Scratch, with `grayscale` derived from the profile's palette.

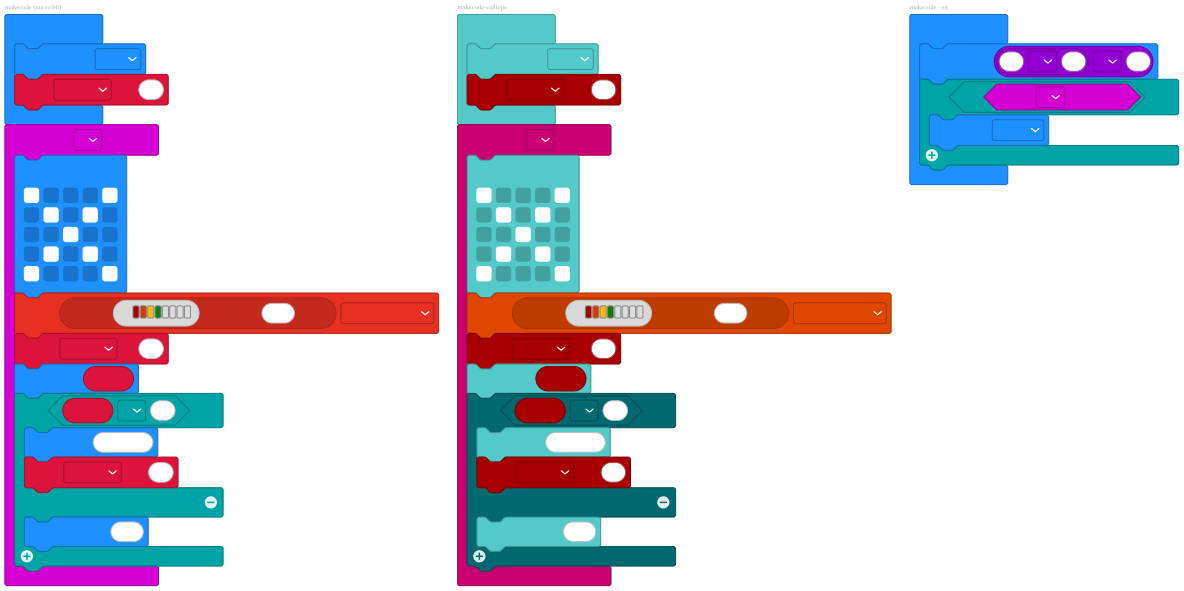
6 MakeCode

6.1 makecode() — Render MakeCode Blocks

`makecode()` renders the blocks of the MakeCode editors for the micro:bit and the Calliope mini. The look is Blockly's zelos renderer as the editors run it — pills and hexagons as tall as their block, white literal pills, the − and + buttons of an if block, MakeCode's monospace labels — and the block texts are the editors' own.

```
#makecode(
  text,
  profile: auto,      // "makecode" (micro:bit), "makecode-calliope"
  language: auto,     // "de" by default
  theme: auto,
  scale: auto,
  font: auto,
  colors: auto,
  line-numbers: auto,
  line-number-start: auto,
  line-number-gutter: auto,
  inset-scale: auto,
)
```

```
#makecode("
beim Start
  zeige Symbol [Herz v]
ende
wenn Knopf [A v] geklickt
  zeige Zahl ((1) + (2))
  pausiere (ms) (100)
ende
")
```



6.2 Profiles

`"makecode"` (the default) is the micro:bit editor, `"makecode-calliope"` the Calliope mini's: the same blocks plus the Calliope's own (motors, the RGB LED), in the Calliope palette. `set-blockst(profile: "makecode-calliope")` sets the default for `makecode()`; `colors:` overrides single categories.

6.3 Writing MakeCode programs

- **Units.** A unit the editor shows in parentheses is typed like a value and drawn as the label: `pausiere (ms) (100)`, `Temperatur (°C)`.
- **Operators** take the editor's glyph or a spelling: `×` or `*`, `÷` or `/`, `≥` or `>=`, `≤` or `<=`, `≠` or `!=`.
- **LED matrix.** 25 cells, `#` lit and `.` dark, in any grouping: `zeige LEDs [#...#|.#.#.|..#..|.#.#.|#...#]`.
- **Melody editor.** Eight notes or rests: `spiele (Melodie [C D E F - - - -] mit Tempo (120) (bpm)) [bis zum Ende v]`.
- **Else.** `ansonsten` / `else` opens the else branch, which gets the editor's – and + buttons.
- **Unknown labels** are drawn as written with the category from a `::kategorie` suffix: `basic`, `input`, `music`, `led`, `radio`, `loops`, `logic`, `variables`, `math`, `functions`, `arrays`, `text`, `game`, `images`, `pins`, `serial`, `control`.

6.4 Languages

The block texts are the editors' own in every language the editors offer: German (`language: "de"`, the default) and English read off the running editors, the other 34 — `"fr"`, `"es"`, `"ja"`, `"zh-cn"`, … — from the translation service the editors load at run time (approved strings only; an untranslated block keeps its English text, as in the editor). A bare code picks the regional variant the editor ships (`"es"` → es-ES, `"pt"` → pt-BR). The end and else markers follow the language (`fin` / `sinon`, □□□ / □□□□□, …); `ende`, `end` and `else` work everywhere.

6.5 Code fences and parsing

`raw-makecode()` renders `makecode` and `microbit` fences with the micro:bit profile and `calliope` fences with the Calliope mini profile. `makecode-parse()` returns the AST.

```
#show: raw-makecode()
#makecode-parse(text, language: "de", profile: "makecode")
```

7 NEPO (Open Roberta)

`nepo()` renders the blocks of Open Roberta Lab for the Calliope mini and the micro:bit — a renderer of its own, contributed by lkoehl. Block text prints in Open Roberta's own wording even when the source used a colloquial alias.

```
#nepo(
  code,
  language: "de",
  platform: "calliope",   // "calliope", "calliopev3", "microbit"
  theme: auto,
  scale: auto,
  font: auto,
)
```

```
#nepo("
Start
  Zeige Text \"Hallo\"
  Wiederhole unendlich oft
    Schalte RGB LED an (#ff0000)
  Ende
")
```



`raw-nepo()` renders `nepo` fences, `nepo-parse()` returns the block tree. The set of blocks is the beginner set of the three platforms, plus the logic, loop, radio and sensor blocks of the lab's expert toolbox: negation, the counting loop with a step, the accelerometer, the radio blocks and the Calliope mini's Grove sensors. Units follow the sensor's mode and ports come from the robot configuration, as in the lab. `examples/nepo` in the repository holds the programme collection, a sheet of every block added after 0.4.0, and the comparison against Open Roberta's own drawings.

8 Labels and line numbers

A line that ends in `#name` is labelled. Labels are collected from every rendered block group — Scratch, Blockly, MakeCode — and can be queried later, so a worksheet can say „the loop in line 3“ and be right after the program changes.

8.1 blockst-labels() — Query Labels

```
#blockst-labels("loop")   // → line number or "NaN"
#blockst-labels()         // → full label → line dictionary
```

8.2 scratch-labels() — Extract Labels

Parses Scratch text and returns a dictionary mapping label names to line numbers, without rendering.

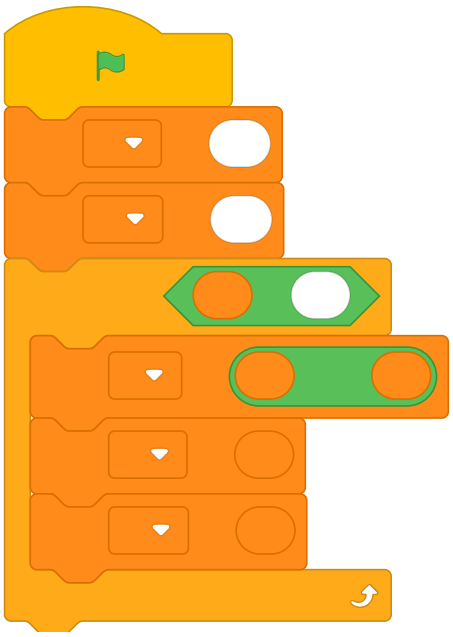
```
#let labels = scratch-labels("
repeat (4) #loop
  move (20) steps #step
  turn cw (90) degrees
end
")
// labels = ("loop": 1, "step": 2)
```

8.3 blockst-register-labels() — Pre-register Labels

Registers labels globally without rendering blocks — for labels that are needed before the first block output.

```
#blockst-register-labels("
repeat (4) #loop
  move (20) steps #step
end
")
```

Euclidean Algorithm (gcd label walkthrough)



Explanation of labeled lines

- **Line 4**: Loop condition `b != 0` controls termination.
- **Line 5**: Core rule: `r = a mod b`.
- **Line 7**: State update that advances to the next pair.

9 Running Scratch programs

blockst includes an execution engine that runs Scratch programs visually — for demonstrating program flow and pen drawing. It runs Scratch text only.

9.1 scratch-run Module

```
#import "@preview/blockst:0.4.2": scratch-run, set-scratch-run

#scratch-run.stage("...", scale: 2)
#scratch-run.grid("...", grid: true)
```

9.1.1 scratch-run.stage() — Stage Canvas

Renders pen drawing on a stage canvas, like the Scratch stage.

```
#scratch-run.stage(
  program,
  language: "en",      // locale for block text
  size: auto,          // (width, height) in pixels (default 480×360)
  scale: auto,         // drawing scale multiplier
  start: auto,         // (x: 0, y: 0, angle: 90)
  pen: auto,           // (down: false, color: ..., size: ...)
  background: auto,    // background color (e.g. white, black)
  cursor: auto,        // show/hide turtle cursor (default: true)
  border: auto,        // show/hide stage border (default: true)
)
```

9.1.2 scratch-run.grid() — Coordinate Grid

Renders the drawing on a Cartesian coordinate grid with optional axes and grid lines.

```
#scratch-run.grid(
  program,
  language: "en",      // locale for block text
  x: auto,             // view bounds (tuple, e.g. (-10, 10))
  y: auto,             // view bounds (tuple)
  step: auto,          // grid step size
  scale: auto,         // drawing scale multiplier
  start: auto,         // (x: 0, y: 0, angle: 90)
  pen: auto,           // (down: false, color: ..., size: ...)
  background: auto,    // background color
  axes: auto,          // show axis lines (default: false)
  grid: auto,          // show grid lines (default: false)
  grid-style: auto,    // grid line stroke (default: 0.5pt + gray)
  cursor: auto,        // show/hide turtle cursor (default: true)
  fit: auto,           // auto-fit view to drawing bounds
)
```

9.1.3 set-scratch-run() — Run Global Defaults

```
#set-scratch-run(
  scale: none,         // default scale for all runs
  start: none,         // (x: 0, y: 0, angle: 90)
  pen: none,           // (down: false, color: ..., size: ...)
  background: none,    // default background color
  cursor: none,        // show/hide turtle cursor (default: true)
```

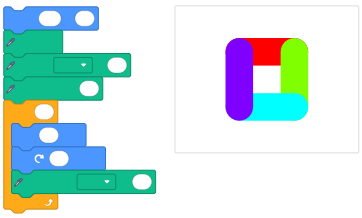
```
  stage: none,          // (size: (480, 360), border: true)
  grid: none,           // (visible: false, axes: false, step: auto, style: auto)
)
```

9.1.4 Supported Execution Commands

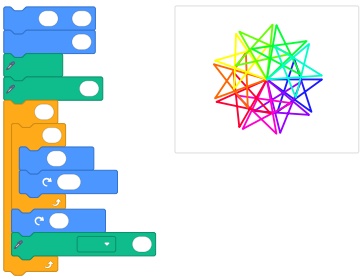
Category	Commands
Motion	`move`, `turn-right`, `turn-left`, `set-direction`, `go-to`, `set-x`, `set-y`, `change-x`, `change-y`
Pen	`pen-down`, `pen-up`, `set-pen-color`, `set-pen-size`, `change-pen-size`, `erase-all`, `stamp`, `set-pen-param`, `change-pen-param`
Variables	`set-variable`, `change-variable`, `variable`
Operators	`plus`, `minus`, `multiply`, `divide`, `modulo`, `random`, `round`, `greater`, `less`, `equals`, `op-and`, `op-or`, `op-not`
Control	`repeat`, `repeat-until`, `if-then`, `if-else`, `wait`
Looks	`say`, `think`
Shapes	`square`, `triangle`, `circle`, `star`, `spiral`
German	`gehe`, `drehe-rechts`, `drehe-links`, `setze-richtung`, `gehe-zu`, `stift-ein`, `stift-aus`, `setze-stiftfarbe-auf`, `setze-stiftdicke`, `setze-variable`, `aendere-variable`, `groesser`, `kleiner`, `gleich`, `und`, `oder`, `nicht`, `mal`, `geteilt`, `zufallszahl`

9.1.5 Complete Example

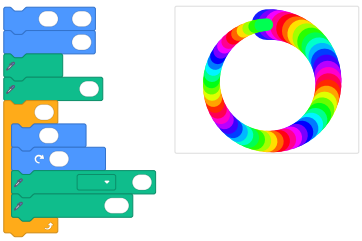
1) Hue Square



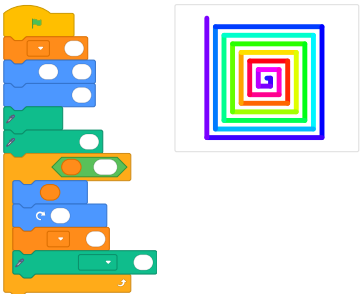
2) Star Rosette (Nested repeat)



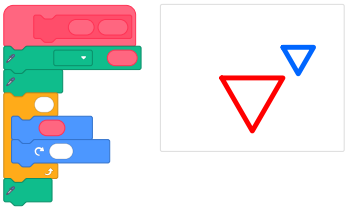
3) Colored Spiral (Hue + pen size)



4) Conditional Loop

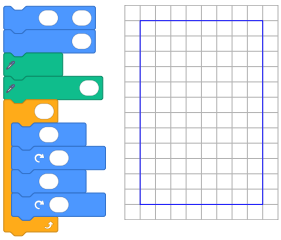


5) Custom Block: Triangle Pattern





6) Grid Preview (Axes + fixed bounds)



```
#let square-program = "
go to x: (-45) y: (45)
pen down
set pen [color v] to (0)
set pen size to (45)
repeat (4)
  move (90) steps
  turn cw (90) degrees
  change pen [color v] by (25)
end"

#set-scratch-run(
  stage: (size: (300, 240)),
  start: (x: 0, y: 0, angle: 90),
)

#grid(
  columns: (auto, auto),
  gutter: 6mm,
  [#scratch(square-program)],
  [#scratch-run.stage(square-program, scale: 2)],
)
```

10 Importing Scratch projects (SB3)

blockst can read real Scratch 3 project files (`.sb3`) and extract scripts, variables, lists, images, and screen previews.

10.1 Basic Workflow

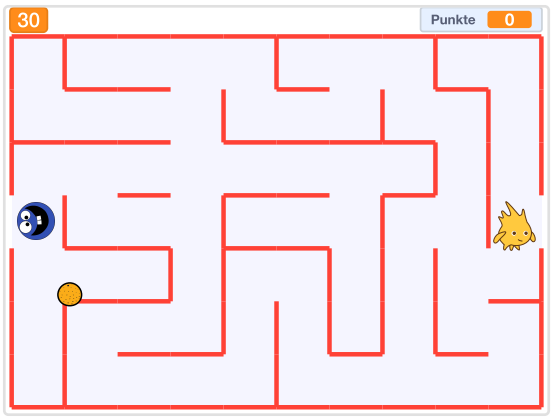
```
#let project = read("my-project.sb3", encoding: none)

#sb3.render-sb3-scripts(project, language: "en", target: "Sprite1")
```

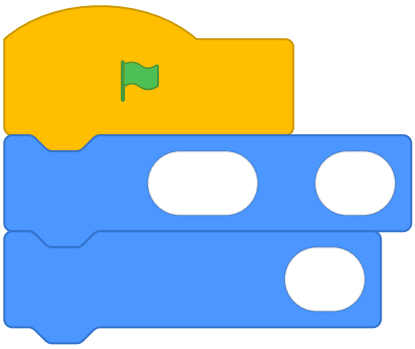
The `.sb3` file must be read as raw bytes (`encoding: none`).

10.2 render-sb3-scripts()

```
#sb3.render-sb3-scripts(
  sb3-bytes,                  // raw .sb3 file bytes (read with encoding: none)
  script-number: auto,        // global script index (1-based)
  target-script-number: auto, // script index within selected target
  target: auto,               // filter by target name ("Stage", sprite name, or auto for
all)
  sb3-plugin: auto,           // WASM plugin path (auto = bundled plugin)
  language: "en",             // locale for block text
  show-headers: auto,         // show target name header
  header-gap: 1.5mm,          // spacing between target name and scripts
  script-gap: 3mm,            // spacing between individual scripts
)
```



2. Sprite: Pacman - Script 2



10.3 render-sb3-variables()

```
#sb3.render-sb3-variables(
  sb3-bytes,                    // raw .sb3 file bytes
  target: auto,                 // filter by target name
  target-variable-name: auto,   // filter by variable name
  target-variable-number: auto,// variable index within target (1-based)
  sb3-plugin: auto,             // WASM plugin path
  language: "en",               // locale for display text
  show-target-headers: auto,    // show target name header
  target-gap: 2mm,              // spacing between targets
  item-gap: 0.8mm,              // spacing between variable items
)
```

10.4 render-sb3-lists()

```
#sb3.render-sb3-lists(
  sb3-bytes,                    // raw .sb3 file bytes
  target: auto,                 // filter by target name
  target-list-name: auto,       // filter by list name
  target-list-number: auto,     // list index within target (1-based)
  sb3-plugin: auto,             // WASM plugin path
  language: "en",               // locale for display text
  show-target-headers: auto,    // show target name header
  target-gap: 2mm,              // spacing between targets
  item-gap: 0.8mm,              // spacing between list items
)
```

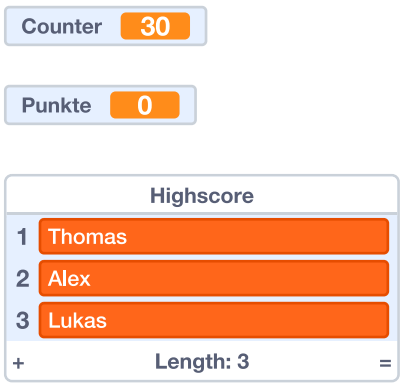
10.5 Standalone Monitors

10.5.1 list-monitor()

```
#list-monitor(
  name: "List",       // display name shown in header
  items: (),          // array of values to display
  width: 5.2cm,       // monitor width
  height: auto,       // auto-grows to fit content
  length-label: auto,// show length indicator (e.g. "length: 3")
)
```

10.5.2 variable-monitor()

```
#variable-monitor(
  name: "Variable",  // display name shown in header
  value: 0,          // current value to display
)
```



10.6 screen-preview()

```
#sb3.sb3-screen-preview(
  sb3-bytes,           // raw .sb3 file bytes
  width: 480,          // stage rendering width in pixels
  height: 360,         // stage rendering height in pixels
  unit: 1,             // size multiplier (2 = double size)
  background: none,    // override background color
  show-border: true,   // show stage border
  show-backdrop: true, // show costume/backdrop image
  monitor-scale: 1.5,  // scale factor for variable/list overlays
  language: auto,      // locale for monitor text
)
```

Renders a static Scratch stage preview with sprites, backdrop, and monitors.

10.7 Image Helpers

```
// List all image assets in the project
#sb3.sb3-image-assets-catalog(sb3-bytes, target: auto)

// Render a single image asset
#sb3.sb3-image(
  sb3-bytes,          // raw .sb3 file bytes
```

```
  target: auto,       // filter by sprite/stage name
  image-number: auto, // global image index (1-based)
  target-image-number: auto, // image index within target (1-based)
  image-name: auto,   // filter by image name (e.g. "costume1")
  width: auto,        // output width (auto = original size)
  height: auto,       // output height
)
```

10.8 Catalog Helpers

```
// Grouped script metadata (targets, scripts, blocks count)
#sb3.sb3-scripts-catalog(sb3-bytes)

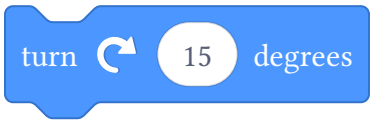
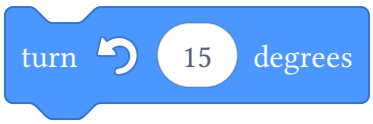
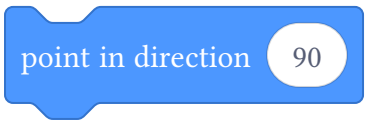
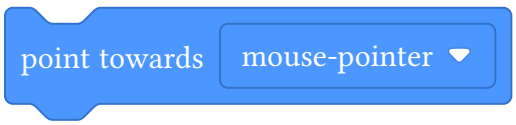
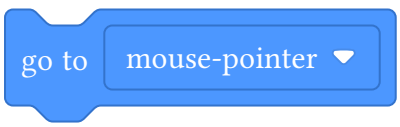
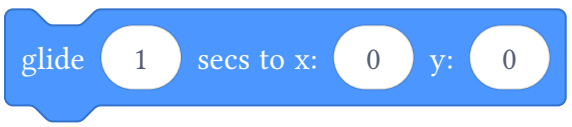
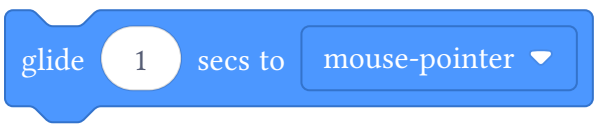
// Target states (variables, lists, and sprite properties)
#sb3.sb3-state-catalog(sb3-bytes)

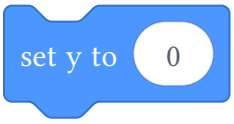
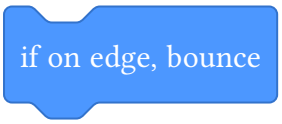
// Convert a specific SB3 script to Scratch text
#sb3.sb3-bytes-to-scratch-text(
  sb3-bytes,          // raw .sb3 file bytes
  script-number: auto,// global script index (1-based)
  language: "en",     // locale for output text
)
```

11 Scratch block catalog

Every block of each Scratch 3 category, rendered live, next to the text that produces it. The Blockly and MakeCode catalogs follow.

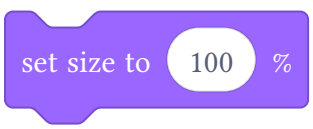
11.1 Motion

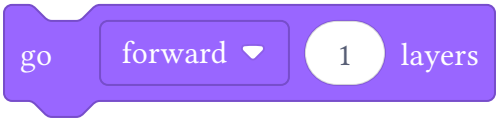
Block	Code
	`move (10) steps`
	`turn cw (15) degrees`
	`turn ccw (15) degrees`
	`point in direction (90)`
	`point towards [mouse-pointer v]`
	`go to x: (0) y: (0)`
	`go to [mouse-pointer v]`
	`glide (1) secs to x: (0) y: (0)`
	`glide (1) secs to [mouse-pointer v]`
	`change x by (10)`

Block	Code
	`set x to (0)`
	`change y by (10)`
	`set y to (0)`
	`if on edge, bounce`
	`set rotation style [left-right v]`
	`(x position)`
	`(y position)`
	`(direction)`

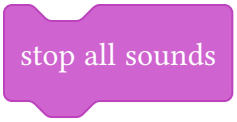
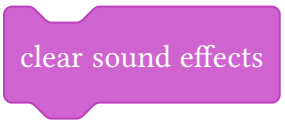
11.2 Looks

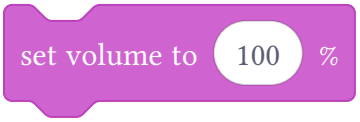
Block	Code
	`say [Hello!] for (2) seconds`
	`say [Hello!]`
	`think [Hmm...] for (2) seconds`
	`think [Hmm...]`

Block	Code
	`show`
	`hide`
	`switch costume to [costume1 v]`
	`next costume`
	`switch backdrop to [backdrop1 v]`
	`next backdrop`
	`change [color v] effect by (25)`
	`set [color v] effect to (0)`
	`clear graphic effects`
	`change size by (10)`
	`set size to (100) %`
	`go to [front v] layer`

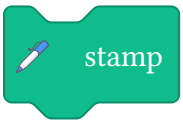
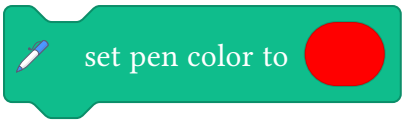
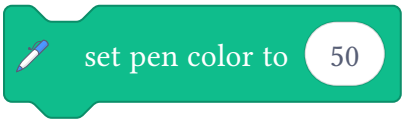
Block	Code
	`go [forward v] (1) layers`
	`(costume [number v])`
	`(backdrop [number v])`
	`(size)`

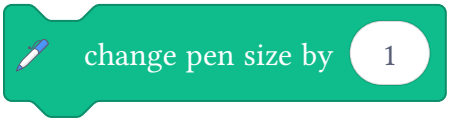
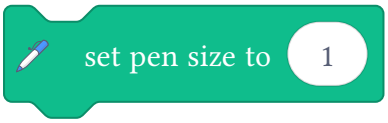
11.3 Sound

Block	Code
	`play sound [pop v] until done`
	`start sound [pop v]`
	`stop all sounds`
	`change [pitch v] effect by (10)`
	`set [pitch v] effect to (100)`
	`clear sound effects`
	`change volume by (-10)`

Block	Code
	`set volume to (100) %`
	`(volume)`

11.4 Pen

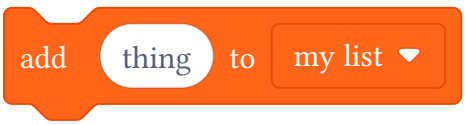
Block	Code
	`erase all`
	`stamp`
	`pen down`
	`pen up`
	`set pen color to [#ff0000]`
	`change pen color by (10)`
	`set pen color to (50)`
	`change pen shade by (10)`
	`set pen shade to (50)`

Block	Code
	`change pen size by (1)`
	`set pen size to (1)`

11.5 Variables

Block	Code
	`set [my variable v] to (0)`
	`change [my variable v] by (1)`
	`show variable [my variable v]`
	`hide variable [my variable v]`
	`(my variable)`

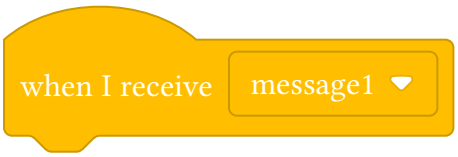
11.6 Lists

Block	Code
	`add [thing] to [my list v]`
	`delete (1) of [my list v]`
	`delete all of [my list v]`

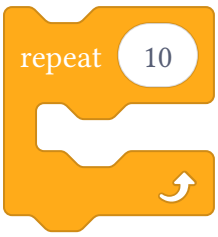
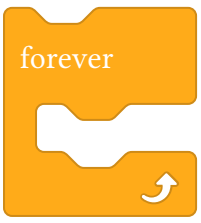
Block	Code
	`insert [thing] at (1) of [my list v]`
	`replace item (1) of [my list v] with [thing]`
	`(item (1) of [my list v])`
	`(item # of [thing] in [my list v])`
	`(length of [my list v])`
	`<[my list v] contains [thing]?>`
	`show list [my list v]`
	`hide list [my list v]`

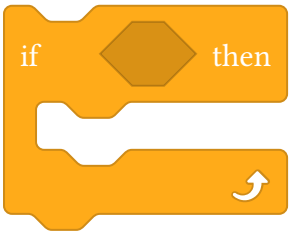
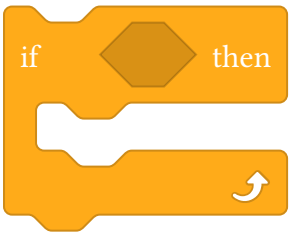
11.7 Events

Block	Code
	`when green flag clicked`
	`when [space v] key pressed`
	`when this sprite clicked`

Block	Code
	`when backdrop switches to [backdrop1 v]`
	`when [loudness v] > (10)`
	`when I receive [message1 v]`
	`broadcast [message1 v]`
	`broadcast [message1 v] and wait`

11.8 Control

Block	Code
	`wait (1) seconds`
	`repeat (10)`
	`end`
	`forever`

Block	Code
	`end`
	`if <> then`
	`end`
	`if <> then`
	`else`
	`end`
	`wait until <>`
	`repeat until <>`
	`end`
	`stop [all v]`

Block	Code
	`when I start as a clone`
	`create clone of [myself v]`
	`delete this clone`

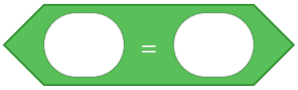
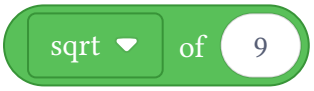
11.9 Sensing

Block	Code
	`<touching [mouse-pointer v]?>`
	`<touching color [#ff0000]?>`
	`<color [#ff0000] is touching [#00ff00]?>`
	`(distance to [mouse-pointer v])`
	`ask [What's your name?] and wait`
	`(answer)`
	`<key [space v] pressed?>`
	`<mouse down?>`
	`(mouse x)`
	`(mouse y)`

Block	Code
	`(loudness)`
	`(timer)`
	`reset timer`
	`([x position v] of [Sprite1 v])`
	`(current [year v])`
	`(days since 2000)`
	`(username)`

11.10 Operators

Block	Code
	`(() + ())`
	`(() - ())`
	`(() * ())`
	`(() / ())`
	`(pick random (1) to (10))`
	`<() > ()>`
	`<() < ()>`

Block	Code
	`<() = ()>`
	`<> and <>`
	`<> or <>`
	`not <>`
	`(join [apple] [banana])`
	`(letter (1) of [apple])`
	`(length of [apple])`
	`([apple] contains [a]?)`
	`(() mod ())`
	`(round ())`
	`([sqrt v] of (9))`

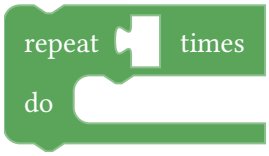
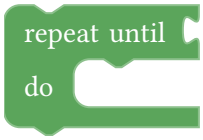
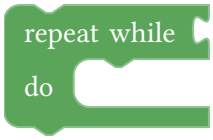
12 Blockly block catalog

Every block the Blockly dialect knows, built from the locale files the parser reads (`scripts/scratchblocks-wasm/data/dialects/locales/`). Empty slots stand for a value `()` or a drop-down `[ v]`; write the content in your own document. The standard blocks are shown in English (`language: "en"`); the same blocks exist in 24 languages, German being the default, see the languages of Blockly and jwinf.

12.1 Standard blocks

The `blockly` profile: Blockly's standard blocks in today's wording.

12.1.1 Schleifen

Block	Code
	`count with [ v] from () to () by ()` `end`
	`for each item [ v] in list ()` `end`
	`repeat () times` `end`
	`repeat until ()` `end`
	`repeat while ()` `end`

12.1.2 Logik

Block	Code
	`if ()` `end`
	`<() = ()>`

Block	Code
	`<() ≥ ()>`
	`<() > ()>`
	`<() ≤ ()>`
	`<() < ()>`
	`<() ≠ ()>`
	`<not ()>`
	`(null)`
	`<() and ()>`
	`<() or ()>`
	`(test ())`

12.1.3 Mathe

Block	Code
	`(() + ())`
	`(() ÷ ())`
	`(() × ())`
	`(() ^ ())`

Block	Code
	`(() - ())`
	`(constrain () low () high ())`
	`(remainder of () ÷ ())`

12.1.4 Text

Block	Code
	`to [ v] append text ()`
	`item`
	`(in text () [ v] ())`
	`<() is empty>`
	`(length of ())`

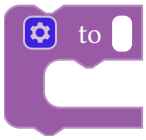
12.1.5 Listen

Block	Code
	`(create empty list)`
	`<() is empty>`
	`(length of ())`
	`(create list with item () repeated () times)`
	`(reverse ())`

12.1.6 Variablen

Block	Code
	`change [ v] by ()`
	`set [ v] to ()`

12.1.7 Funktionen

Block	Code
	`to ()` `end`
	`if () return ()`

12.2 jwinf

The `jwinf` profile: the robot and turtle world blocks of jwinf.de, plus the standard blocks whose wording differs there. jwinf is a German competition, its world blocks exist in German only.

12.2.1 Start

Block	Code
	`Programm`
	`Roboter-Programm`
	`Schildkröten-Programm`

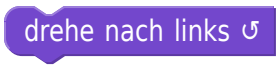
12.2.2 Aktionen

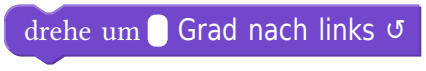
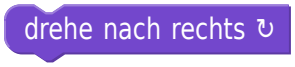
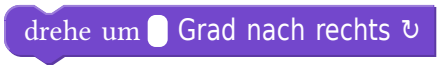
Block	Code
	`gehe rückwärts`
	`lege Objekt ab`
	`gehe nach rechts`
	`gehe`
	`(links vom Gitterrand)`

Block	Code
	`(rechts vom Gitterrand)`
	`springe`
	`drehe nach links`
	`gehe nach oben`
	`(Hindernis rechts)`
	`(vor Hindernis)`
	`(Hindernis oben)`
	`(Hindernis unten)`
	`(Hindernis links)`
	`(auf Kiste)`
	`(sur un lanceur laser)`
	`(auf Zahl)`
	`(auf Objekt)`
	`(auf Tafel)`
	`(Plattform darüber)`
	`(vor Plattform)`
	`schiebe Kiste`
	`(vor Kiste)`
	`(Zahl auf dem Feld)`
	`drehe nach rechts`

Block	Code
	`Rückkehr von der Schießrichtung ()`
	`schieße Laser in Richtung ()`
	`gehe nach unten`
	`drehe um`
	`gehe nach links`
	`hebe Objekt auf`
	`schreibe Zahl ()`

12.2.3 Schildkroete

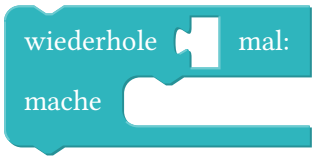
Block	Code
	`(turtle's column)`
	`setze Farbe ()`
	`gehe () Schritte`
	`gehe () Schritte zurück`
	`setze Stift auf`
	`hebe Stift ab`
	`(turtle's row)`
	`drehe (Grad) ()`
	`drehe um ()° nach [ v]`
	`drehe um () nach [ v]`
	`drehe nach links ↺`

Block	Code
	`drehe um ()° nach links ↺`
	`drehe um () nach links ↺`
	`drehe um () Grad nach links ↺`
	`drehe nach rechts ↻`
	`drehe um ()° nach rechts ↻`
	`drehe um () nach rechts ↻`
	`drehe um () Grad nach rechts ↻`

12.2.4 TurtleInput

Block	Code
	`(Eingabewert)`

12.2.5 Schleifen

Block	Code
	`wiederhole () mal:` `ende`

12.2.6 Text

Block	Code
	`etwas`

12.2.7 Listen

Block	Code
	`(erzeuge Liste mit () mal dem Element ())`

12.2.8 Debug

Block	Code
	`messagebox ()`
	`logge ()`

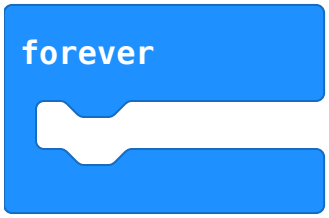
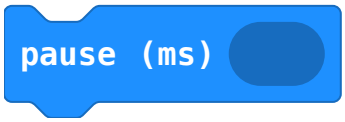
13 MakeCode block catalog

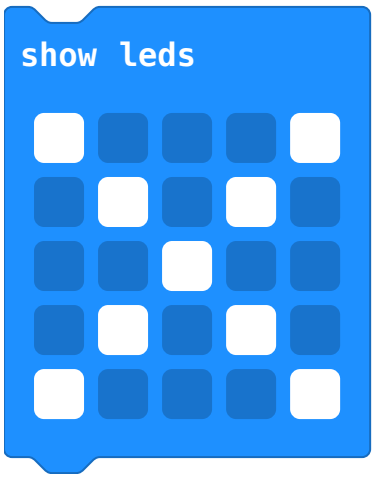
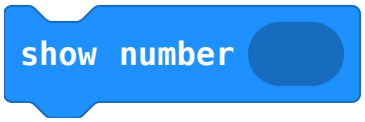
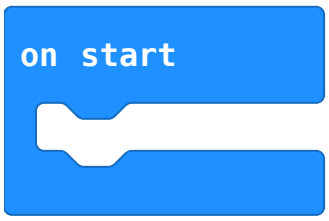
Every block of the MakeCode editors, built from the locale files the parser reads, shown in English (`language: "en"`). The same blocks exist in all 36 editor languages, German being the default, see the languages of MakeCode.

13.1 micro:bit

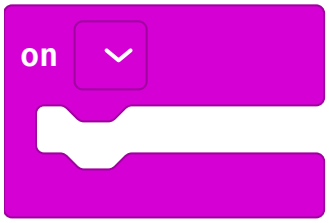
The `makecode` profile: the blocks of makecode.microbit.org.

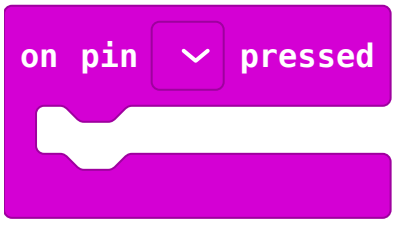
13.1.1 Basic

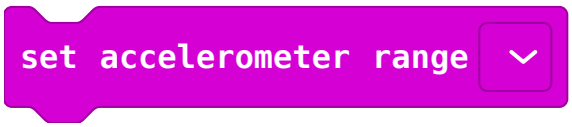
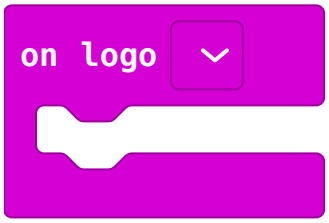
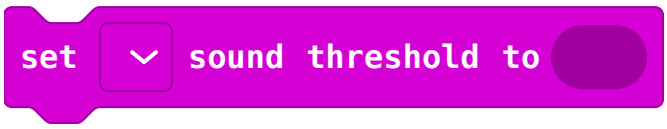
Block	Code
	`show arrow ()`
	`show icon [ v]`
	`clear screen`
	`forever` `end`
	`pause (ms) ()`
	`show string ()`

Block	Code
	`show leds [#...#|.#.#.|..#..|.#.#.|#...#]`
	`show number ()`
	`on start` `end`

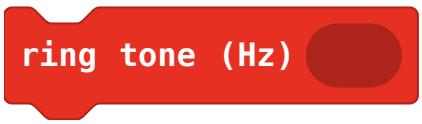
13.1.2 Input

Block	Code
	`(acceleration (mg) [ v])`
	`on button [ v] pressed` `end`
	`on [ v]` `end`

Block	Code
	`<button [ v] is pressed>`
	`(light level)`
	`(magnetic force (µT) [ v])`
	`(rotation (°) [ v])`
	`(running time (ms))`
	`(running time (micros))`
	`(sound level)`
	`(compass heading (°))`
	`on pin [ v] pressed` `end`
	`<pin [ v] is pressed>`
	`on pin [ v] released` `end`

Block	Code
	`set accelerometer range [ v]`
	`(temperature (°C))`
	`<is [ v] gesture>`
	`calibrate compass`
	`on logo [ v]` `end`
	`<logo is pressed>`
	`set [ v] sound threshold to ()`

13.1.3 Music

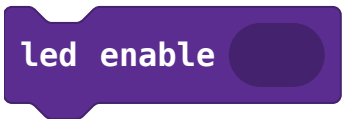
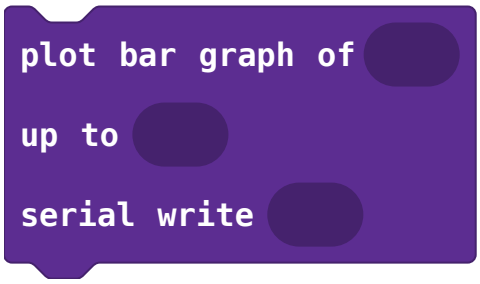
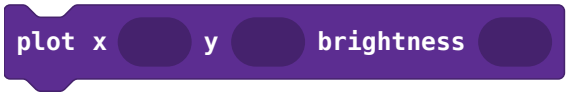
Block	Code
	`([ v] beat)`
	`change tempo by (bpm) ()`
	`rest for ()`
	`ring tone (Hz) ()`

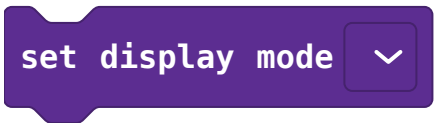
Block	Code
	`set tempo to (bpm) ()`
	`stop melody [ v]`
	`(tempo (bpm))`
	`music on [ v]` `end`
	`play () [ v]`
	`set built-in speaker ()`
	`<sound is playing>`
	`stop all sounds`
	`(melody [C D E F - - - -] at tempo () (bpm))`

	`(() start frequency () end frequency ()`
	`duration () start volume () end volume ()`
	`effect [ v] interpolation [ v])`

	`set volume ()`

13.1.4 Led

Block	Code
	`(brightness)`
	`led enable ()`
	`toggle x () y ()`
	`plot x () y ()`
	`plot bar graph of () up to () serial write ()`
	`plot x () y () brightness ()`
	`<point x () y ()>`
	`(point x () y () brightness)`
	`set brightness ()`
	`stop animation`
	`unplot x () y ()`

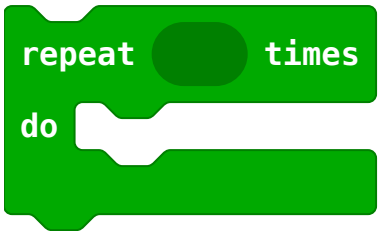
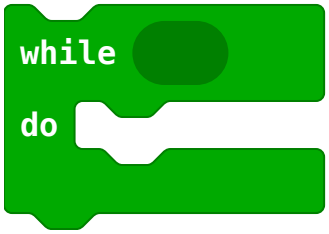
Block	Code
	`set display mode [ v]`

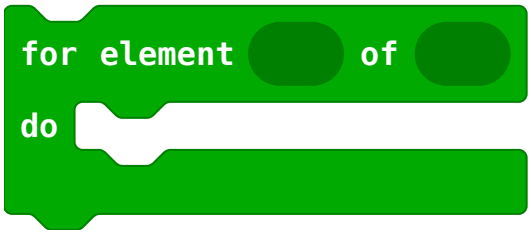
13.1.5 Radio

Block	Code
	`radio send ()`
	`on radio () received` `end`
	`radio send number ()`
	`radio send string ()`
	`radio send value () = ()`
	`on radio received ()` `end`
	`on radio received () ()` `end`
	`(received packet ())`

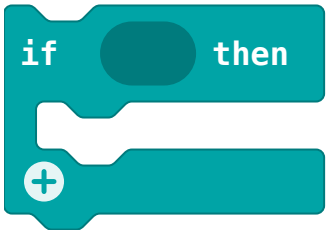
Block	Code
	`radio set frequency band ()`
	`radio set group ()`
	`radio set transmit power ()`
	`radio set transmit serial number ()`

13.1.6 Loops

Block	Code
	`break`
	`continue`
	`repeat () times` `end`
	`while ()` `end`
	`every () ms` `end`

Block	Code
	`for () from 0 to ()` `end`
	`for element () of ()` `end`
	`pause until ()`

13.1.7 Logic

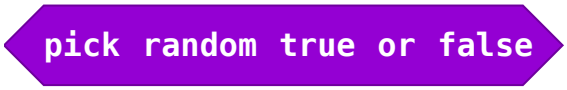
Block	Code
	`if () then` `end`
	`<false>`
	`<true>`
	`<() = ()>`
	`<() ≥ ()>`
	`<() > ()>`
	`<() ≤ ()>`

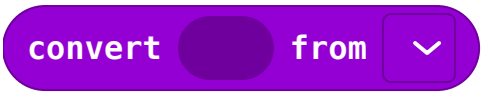
Block	Code
	`<() < ()>`
	`<() ≠ ()>`
	`<not ()>`
	`<() and ()>`
	`<() or ()>`

13.1.8 Variables

Block	Code
	`change [ v] by ()`
	`set [ v] to ()`

13.1.9 Math

Block	Code
	`(pick random () to ())`
	`<pick random true or false>`
	`(() + ())`
	`(() / ())`

Block	Code
	`(() × ())`
	`(() ** ())`
	`(() - ())`
	`(constrain () between () and ())`
	`(convert () from [ v])`
	`(map () from low () high () to low () high ())`
	`(remainder of () / ())`
	`([ v] of () and ())`
	`(absolute of ())`

13.1.10 Functions

Block	Code
	`return ()`
	`call function`
	`function`

13.1.11 Arrays

Block	Code
	`(() find index of ())`
	`() insert at () value ()`
	`(get random value from ())`
	`(get and remove last value from ())`
	`remove last value from ()`
	`() add value () to end`
	`(() get and remove value at ())`
	`() remove value at ()`
	`reverse ()`
	`(get and remove first value from ())`
	`remove first value from ()`
	`(() insert () at beginning)`
	`(empty array)`

Block	Code
	`(() get value at ())`
	`() set value at () to ()`
	`(length of array ())`

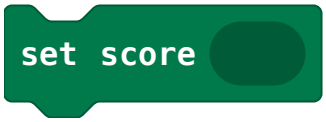
13.1.12 Text

Block	Code
	`(text from char code ())`
	`(char code from () at ())`
	`(compare () to ())`
	`(char from () at ())`
	`<() includes ()>`
	`<() is empty>`
	`(parse to number ())`
	`(split () at ())`
	`(substring of () from () of length ())`
	`(join () ())`

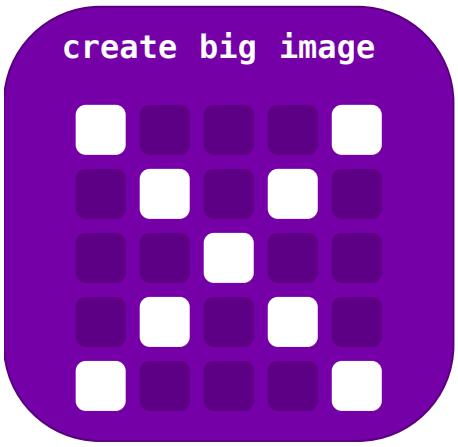
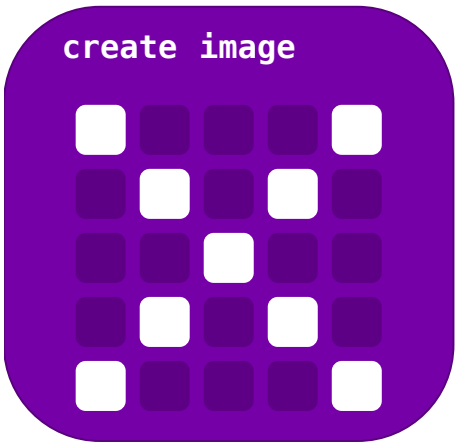
Block	Code
	`(length of ())`
	`(convert () to text)`

13.1.13 Game

Block	Code
	`add life ()`
	`change score by ()`
	`(create sprite at x: () y: ())`
	`delete ()`
	`game over`
	`<is game over>`
	`<is paused>`
	`<is running>`
	`() move by ()`
	`pause`

Block	Code
	`remove life ()`
	`(score)`
	`set life ()`
	`set score ()`
	`() if on edge, bounce`
	`() change [ v] by ()`
	`<is () deleted>`
	`() set [ v] to ()`
	`<is () touching edge>`
	`<is () touching ()>`
	`start countdown (ms) ()`
	`() turn [ v] by (°) ()`

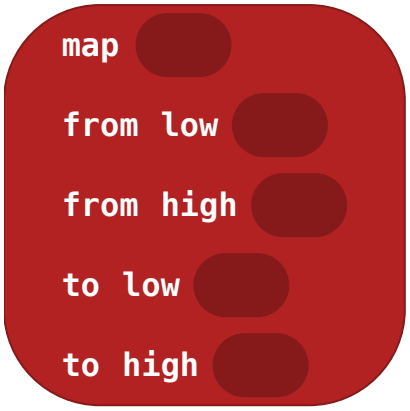
13.1.14 Images

Block	Code
	`(arrow image [ v])`
	`(icon image [ v])`
	`(create big image [#...#|.#.#.|..#..|.#.#.|#...#])`
	`(create image [#...#|.#.#.|..#..|.#.#.|#...#])`
	`scroll image () with offset () and interval (ms) ()`
	`show image () at offset () and interval (ms) ()`

13.1.15 Pins

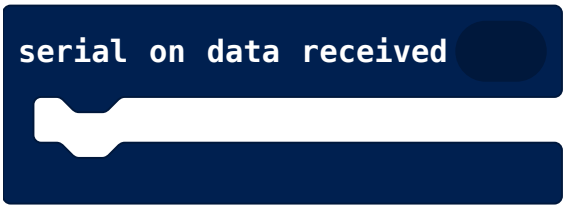
Block	Code
	`(analog pin [ v])`

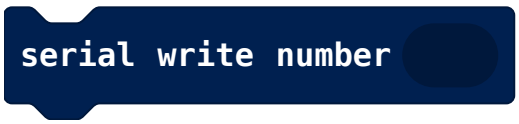
Block	Code
	`analog pitch () for (ms) ()`
	`analog set pitch pin ()`
	`(analog read pin ())`
	`(digital read pin ())`
	`analog set period pin () to (µs) ()`
	`analog write pin () to ()`
	`digital write pin () to ()`
	`set pin () to emit [ v] events`
	`set pull pin () to [ v]`
	`servo write pin () to ()`
	`servo set pulse pin () to (µs) ()`
	`set [ v] to touch mode [ v]`
	`(digital pin [ v])`

Block	Code
	`i2c write number at address () with value () of format [ v] repeated ()`
	`(map () from low () from high () to low () to high ())`
	`neopixel matrix width pin () ()`
	`set audio pin ()`
	`set audio pin enabled ()`
	`(i2c read number at address () of format [ v] repeated ())`
	`on pin [ v] pulsed [ v]` `end`
	`(pulse duration (µs))`
	`(pulse in (µs) pin () pulsed [ v])`

Block	Code
	`spi format bits () mode ()`
	`spi frequency ()`
	`spi set pins MOSI () MISO () SCK ()`
	`(spi write ())`

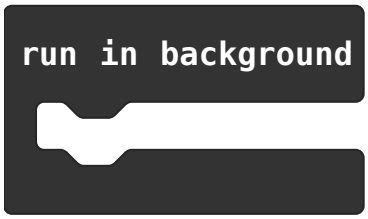
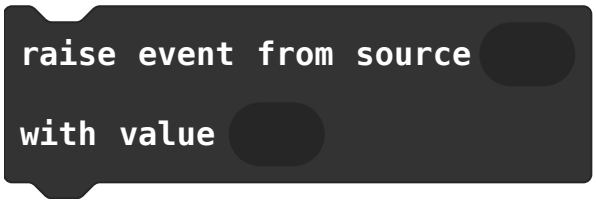
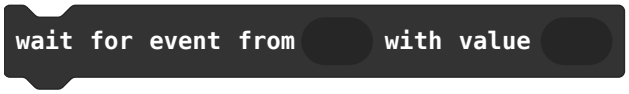
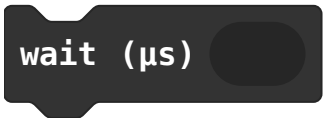
13.1.16 Serial

Block	Code
	`serial set rx buffer size to ()`
	`serial set tx buffer size to ()`
	`serial set write line padding to ()`
	`serial on data received ()` `end`
	`(serial read string)`
	`(serial read line)`
	`(serial read until ())`
	`(serial read buffer ())`

Block	Code
	`serial redirect to TX [ v] RX [ v] at baud rate [ v]`
	`serial redirect to USB`
	`serial set baud rate [ v]`
	`serial write buffer ()`
	`serial write line ()`
	`serial write number ()`
	`serial write numbers ()`
	`serial write string ()`
	`serial write value () = ()`

13.1.17 Control

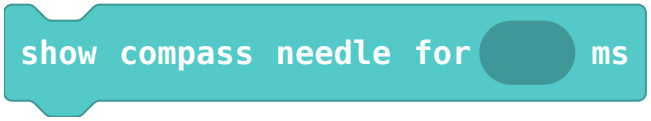
Block	Code
	`(device name)`
	`(device serial number)`
	`(event timestamp)`

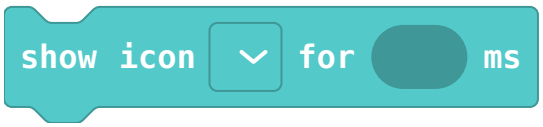
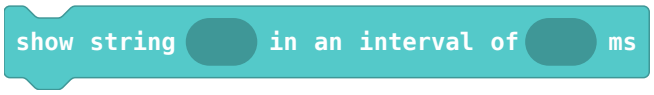
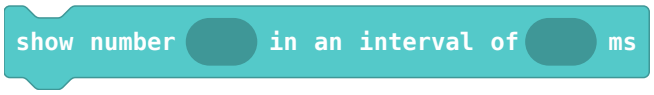
Block	Code
	`(event value)`
	`run in background` `end`
	`on event from () with value ()` `end`
	`raise event from source () with value ()`
	`reset`
	`(millis (ms))`
	`wait for event from () with value ()`
	`wait (µs) ()`

13.2 Calliope mini

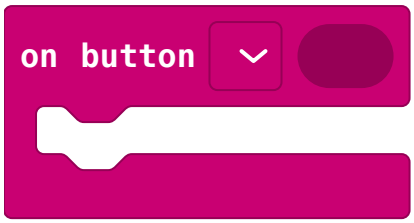
The `makecode-calliope` profile adds the Calliope mini's own blocks and recolours the shared ones; listed here are only the blocks the Calliope editor adds or words differently.

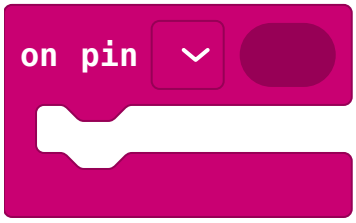
13.2.1 Basic

Block	Code
	`show compass needle for () ms`

Block	Code
	`show icon [ v] for () ms`
	`(red () green () blue ())`
	`show string () in an interval of () ms`
	`set LED to ()`
	`set LED to () () () brightness ()`
	`show number () in an interval of () ms`
	`turn built-in LED off`

13.2.2 Input

Block	Code
	`on button [ v] ()` `end`
	`(rotation (°) [ v])`
	`(running time (micros))`

Block	Code
	`on pin [ v] ()` `end`
	`(sound level)`

13.2.3 Music

Block	Code
	`ring tone (Hz) ()`
	`set tempo to (bpm) ()`

13.2.4 Led

Block	Code
	`led enable ()`
	`toggle x () y ()`

13.2.5 Functions

Block	Code
	`call tuWas`

13.2.6 Arrays

Block	Code
	`() remove value at ()`

Block	Code
	`(empty array)`

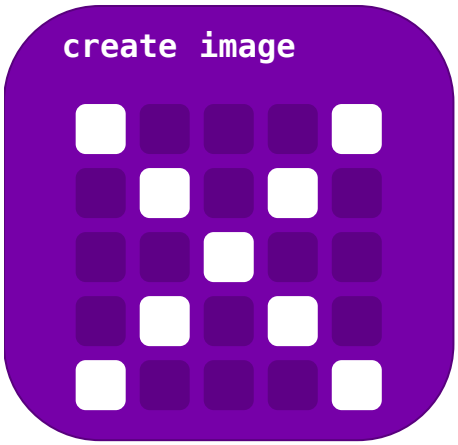
13.2.7 Text

Block	Code
	`(parse to number ())`

13.2.8 Game

Block	Code
	`<is () touching edge>`
	`<is () touching ()>`

13.2.9 Images

Block	Code
	`(create image [#...#|.#.#.|..#..|.#.#.| #...#])`
	`show image () at offset () and interval (ms) ()`

13.2.10 Pins

Block	Code
	`(analog pin [ v])`

Block	Code
	`analog set pitch pin ()`
	`servo set pulse pin () to (µs) ()`
	`(digital pin [ v])`
	`I²C write number at address () with value () of format [ v] repeated ()`
	`(I²C read number at address () of format [ v] repeated ())`
	`(pulse in (µs) pin () pulsed [ v])`

13.2.11 Serial

Block	Code
	`serial set rx buffer size to ()`
	`serial set tx buffer size to ()`
	`serial redirect to USB`
	`serial write numbers ()`

13.2.12 Motors

Block	Code
	`motor [ v] at () %`

14 Contributing

Contributions are welcome: bug reports, missing blocks, parser improvements, rendering polish, docs, and new localizations.

- **Repository:** github.com/Loewe1000/blockst
- **License:** MIT

15 API Reference

The reference is generated from the `///` comments in the source. It covers the public rendering API in `libs/scratch/api.typ`, which `lib.typ` and `package.typ` re-export unchanged.

15.1 `blockly`

```
blockly(
  text,
  profile: auto,
  language: auto,
  theme: auto,
  scale: auto,
  font: auto,
  line-numbering: auto,
  line-numbers: auto,
  line-number-start: auto,
  line-number-first-block: auto,
  line-number-gutter: auto,
  inset-scale: auto,
  colors: auto,
)
```

Render Blockly blocks from text. Same notation as `scratch()`, drawn with Blockly's shapes — notch, puzzle tab, boxed fields — and the vocabulary, categories and colours of a profile:

- `"blockly"` (the default): today's flat look, Blockly's German wording
- `"blockly-klassisch"`: the pre-2019 look
- `"jwinf"`: jwinf.de — classic geometry, the robot and turtle world blocks, and the palette of the robot training tasks
- `"jwinf-turtle"`: the same with the colours of the Freie Turtle-Umgebung

`colors: (logik: "#73cc47")` overrides single categories of any profile.

Labels that no profile knows are drawn as written, with the category taken from a `::kategorie` suffix — on jwinf the normal case, because the same block reads differently from task to task.

```
#blockly("wiederhole (4) mal:\n  gehe nach rechts\nende", profile: "jwinf")
```

Name	Vorgabe
`text`	–
`profile`	`auto`
`language`	`auto`
`theme`	`auto`
`scale`	`auto`
`font`	`auto`
`line-numbering`	`auto`
`line-numbers`	`auto`
`line-number-start`	`auto`
`line-number-first-block`	`auto`
`line-number-gutter`	`auto`
`inset-scale`	`auto`

Name	Vorgabe
`colors`	`auto`

15.2 `blockly-parse`

```
blockly-parse(text, language: "de", profile: "blockly")
```

Parse Blockly text to AST (for programmatic use).

Name	Vorgabe
`text`	–
`language`	`"de"`
`profile`	`"blockly"`

15.3 `blockst`

```
blockst(
  theme: auto,
  scale: auto,
  font: auto,
  line-numbering: auto,
  line-numbers: auto,
  line-number-start: auto,
  line-number-first-block: auto,
  line-number-gutter: auto,
  inset-scale: auto,
  language: auto,
  colors: auto,
  spacing: 1.5em,
  body,
)
```

Optional container for grouped blocks with theme/scale override. Only needed when a specific group should differ from global settings.

Name	Vorgabe
`theme`	`auto`
`scale`	`auto`
`font`	`auto`
`line-numbering`	`auto`
`line-numbers`	`auto`
`line-number-start`	`auto`
`line-number-first-block`	`auto`
`line-number-gutter`	`auto`
`inset-scale`	`auto`
`language`	`auto`
`colors`	`auto`
`spacing`	`1.5em`
`body`	–

15.4 `blockst-labels`

```
blockst-labels(name)
```

Read globally collected line labels from all previously rendered `scratch()` blocks. With a name: `#blockst-labels("start", default: "?")`. Without a name: returns the full label-to-line dictionary.

15.5 `blockst-register-labels`

```
blockst-register-labels(text, language: "en")
```

Register labels globally without rendering blocks. Useful when labels are needed before the first `#scratch()` output appears.

Name	Vorgabe
`text`	–
`language`	`"en"`

15.6 `makecode`

```
makecode(
  text,
  profile: auto,
  language: auto,
  theme: auto,
  scale: auto,
  font: auto,
  line-numbering: auto,
  line-numbers: auto,
  line-number-start: auto,
  line-number-first-block: auto,
  line-number-gutter: auto,
  inset-scale: auto,
  colors: auto,
)
```

Render MakeCode blocks from text — the micro:bit and Calliope mini editors' look: Blockly's zelos renderer with MakeCode's monospace labels. Same notation as `scratch()`; `(…)` is a value, `[… v]` a dropdown, `<…>` a boolean, `ende` / `end` closes a C-block and `sonst` / `else` opens the else branch. The profile picks the target: `"makecode"` (micro:bit, the default) or `"makecode-calliope"`; the language is any of the editors' 36 (`"de"` by default, `"en"`, `"fr"`, `"es"`, `"zh-cn"`, …), with the block texts the editors show in that language. The end marker follows the language (`ende`, `end`, `fin`, …); `end` and `ende` work in every language.

```
#makecode("beim Start\n  zeige Zahl (0)\nende")
```

Name	Vorgabe
`text`	–
`profile`	`auto`
`language`	`auto`
`theme`	`auto`
`scale`	`auto`
`font`	`auto`
`line-numbering`	`auto`
`line-numbers`	`auto`

Name	Vorgabe
`line-number-start`	`auto`
`line-number-first-block`	`auto`
`line-number-gutter`	`auto`
`inset-scale`	`auto`
`colors`	`auto`

15.7 `makecode-parse`

```
makecode-parse(text, language: "de", profile: "makecode")
```

Parse MakeCode text to AST (for programmatic use).

Name	Vorgabe
`text`	–
`language`	`"de"`
`profile`	`"makecode"`

15.8 `raw-blockly`

```
raw-blockly(..args)
```

Enable Blockly code blocks in raw text:

````
#show: raw-blockly()
```blockly
wiederhole (4) mal:
ende
```
````

A `jwinf` fence uses the jwinf profile whatever the arguments say, a `jwinf-turtle` fence the turtle sandbox's colours.

15.9 `raw-makecode`

```
raw-makecode(..args)
```

Enable MakeCode code blocks in raw text:

````
#show: raw-makecode()
```makecode
beim Start
  zeige Zahl (0)
ende
```
````

A `microbit` fence is the same; a `calliope` fence uses the Calliope mini profile whatever the arguments say.

15.10 `raw-scratch`

```
raw-scratch(..args)
```

Enable scratch code blocks in raw text:

```
#show: raw-scratch()
```

With language: `#show: raw-scratch(language: "de")`.

15.11 `scratch`

```
scratch(
  text,
  language: auto,
  theme: auto,
  scale: auto,
  font: auto,
  line-numbering: auto,
  line-numbers: auto,
  line-number-start: auto,
  line-number-first-block: auto,
  line-number-gutter: auto,
  inset-scale: auto,
  colors: auto,
)
```

Render scratch blocks from text. Works standalone or inside `#blockst[...]`. Supports 26 languages: en, de, fr, es, it, pt, nl, pl, ru, ja, ...

Name	Vorgabe
`text`	–
`language`	`auto`
`theme`	`auto`
`scale`	`auto`
`font`	`auto`
`line-numbering`	`auto`
`line-numbers`	`auto`
`line-number-start`	`auto`
`line-number-first-block`	`auto`
`line-number-gutter`	`auto`
`inset-scale`	`auto`
`colors`	`auto`

15.12 `scratch-execute`

```
scratch-execute(text, language: "en")
```

Execute scratch text, producing scratch-run commands.

Name	Vorgabe
`text`	–
`language`	`"en"`

15.13 `scratch-labels`

```
scratch-labels(text, language: "en")
```

Parse scratch text and return a dictionary mapping `#labels` to rendered line numbers.

Name	Vorgabe
`text`	–

Name	Vorgabe
`language`	`"en"`

15.14 `scratch-parse`

```
scratch-parse(text, language: "en")
```

Parse scratch text to AST (for programmatic use).

Name	Vorgabe
`text`	–
`language`	`"en"`

15.15 `set-blockst`

```
set-blockst(
  theme: none,
  profile: none,
  colors: none,
  scale: none,
  stroke-width: none,
  font: none,
  line-numbering: none,
  line-numbers: none,
  line-number-start: none,
  line-number-first-block: none,
  line-number-gutter: none,
  inset-scale: none,
  language: none,
)
```

Global settings applied to all scratch() and sb3 calls.

`colors` lays the document's own category colours over the profile's palette: `set-blockst(colors: (logik: "#73cc47"))` — a hex string or a Typst colour per category. The shades a theme derives (bevel, stroke, high-contrast, grayscale) follow the new fill.

Name	Vorgabe
`theme`	`none`
`profile`	`none`
`colors`	`none`
`scale`	`none`
`stroke-width`	`none`
`font`	`none`
`line-numbering`	`none`
`line-numbers`	`none`
`line-number-start`	`none`
`line-number-first-block`	`none`
`line-number-gutter`	`none`
`inset-scale`	`none`
`language`	`none`