Noteworthy Framework

Examples & *Documentation* (*Solutions*)

Written by :

Sihoo Lee / *Lee Hojun*

NOTEWORTHY

2025-12-28

Preface

Welcome to the **Noteworthy Framework**. This document serves as both a demonstration of the framework's capabilities and a reference for its features.

1 About Noteworthy

Noteworthy is a modular framework for creating beautiful educational documents in Typst. It provides a comprehensive set of tools for:

- **Structured Layouts**: Automated chapters, sections, and covers.
- **Themed Components**: Pre-styled blocks for definitions, theorems, examples, and more.
- **Advanced Plotting**: Integrated 2D and 3D plotting capabilities.
- **Customizable Themes**: A robust theming engine with multiple built-in presets.

2 Using This Guide

Each section of this document demonstrates a specific module of the framework. You can find the source code for these examples in the `content/` directory, which serves as a practical reference for your own documents.

Sihoo Lee & *Lee Hojun*
Noteworthy

Table of Contents

Chapter 04 *Canvas Module*

Chapter 05 *Data Module*

Chapter 06 *Cover Module*

Chapter 07 *Layout Module*

Chapter 08 *Combi Module*

Chapter 00

Architecture & Modules

Understanding Noteworthy's modular structure and file organization.

Chapter 00.01

Introduction

1 Welcome to Noteworthy

Noteworthy is a powerful Typst framework for creating beautiful educational documents with rich content blocks and visualization tools.

1.1 What is Noteworthy?

Definition | Noteworthy

A modular Typst template system designed for creating professional educational materials, textbooks, and technical documentation.

1.2 Key Features

Note | Modular Architecture

Noteworthy is organized into **6 modules**, each handling a specific aspect of document creation:

- **Block** — Semantic content containers (definitions, theorems, proofs)
- **Geometry** — 2D geometric primitives (points, lines, circles)
- **Canvas** — Rendering canvases for plots and visualizations
- **Data** — Tables, data series, and curve interpolation
- **Cover** — Document covers and title pages
- **Layout** — Page layouts and table of contents

1.3 How to Use This Guide

This documentation is organized by module. Each chapter covers one module:

1. **Chapter 0** — Architecture & file structure (you are here)
2. **Chapter 1** — Block module for content containers
3. **Chapter 2** — Geometry module for 2D shapes
4. **Chapter 3** — Canvas module for plotting
5. **Chapter 4** — Data module for tables and series
6. **Chapter 5** — Cover & Layout for document structure

Theorem | Getting Started

Every content file starts with one import:

```
#import "../../templates/templater.typ": *
```

This single import gives you access to all modules.

Chapter 00.02

File Structure

1 File Structure

Understanding the project layout helps you navigate and extend Noteworthy.

1.1 Project Root

Notation | Directory Legend

- 📁 = Directory
- 📄 = File

```
noteworthy/
├── 📁 config/          # Configuration files
│   ├── hierarchy.json  # Chapter/page structure
│   ├── metadata.json   # Title, authors, etc.
│   ├── constants.json  # Display settings
│   └── schemes/        # Color themes
├── 📁 content/         # Your document pages
│   ├── 0/, 1/, 2/...   # Chapter folders
│   └── images/         # Embedded images
├── 📁 templates/       # The template system
│   ├── templater.typ   # Main entry point
│   ├── core/           # Core utilities
│   └── module/         # Feature modules
└── output.pdf          # Compiled document
```

1.2 Templates Directory

The `templates/` folder contains the template system:

Definition | templater.typ

The single entry point that re-exports all modules. Content files only need to import this one file.

Definition | core/

Core utilities shared across all modules:
- `setup.typ` — Configuration loading and theme definition
- `scheme.typ` — Color scheme management
- `parser.typ` — Content parsing for builds

- `scanner.typ` — Content discovery

Definition | module/

Feature modules, each in its own folder with a `mod.typ` entry point:
- `block/` — Content blocks
- `geometry/` — 2D primitives
- `canvas/` — Plotting canvases
- `data/` — Tables and data
- `cover/` — Document covers
- `layout/` — Page layouts

1.3 Module Pattern

Each module follows the same pattern:

Example | Module Structure

```
module/block/
├── mod.typ      # Entry point (exports themed wrappers)
└── block.typ    # Implementation
```

The `mod.typ` file imports the implementation, applies theming, and exports ready-to-use functions.

Chapter 00.03

Module Overview

1 Module Overview

A quick reference for all seven Noteworthy modules.

1.1 The Seven Modules

Definition | block

Semantic content containers.

Key exports:

- `definition`, `theorem`, `proof`
- `example`, `solution`
- `note`, `notation`, `equation`

Definition | shape

2D geometric primitives.

Key exports:

- `point`, `line`, `circle`
- `polygon`, `angle`
- `midpoint`, `intersect-ll`

Definition | graph

Functions and vectors.

Key exports:

- `graph`, `func`, `parametric`
- `vec`, `vec-add`, `vec-project`
- `polar-func`

Definition | canvas

Rendering canvases.

Key exports:

- `cartesian-canvas`
- `polar-canvas`, `trig-canvas`
- `space-canvas`, `graph-canvas`

Definition | data

Data visualization and tables.

Key exports:

- `table-plot`, `value-table`
- `data-series`, `csv-series`
- `curve-through`, `smooth-curve`

Definition | cover

Document covers and title pages.

Key exports:

- `cover`, `chapter-cover`
- `preface`, `project`

Definition | layout

Page layouts and table of contents.

Key exports:

- `outline`

1.2 How Modules Work Together

Note | Typical Workflow

1. Use **block** module to structure your content
2. Use **shape** module to create geometric objects
3. Use **graph** module for functions and vectors
4. Use **canvas** module to render shapes and graphs
5. Use **data** module for tables and data plots
6. **Cover** and **Layout** modules handle document structure

Chapter 01

Block Module

Semantic content blocks for educational documents.

Chapter 01.01

Block Fundamentals

1 Block Fundamentals

The Block module provides semantic content containers for educational documents.

1.1 What is a Block?

Definition | Block

A styled container that gives semantic meaning to content. Blocks help readers identify the type of information they're reading.

1.2 Block Syntax

All blocks follow the same pattern:

```
#blockname("Optional Title")[
  Content goes here...
]
```

Some blocks (like `proof` and `solution`) don't require a title:

```
#proof[
  Content without a title...
]
```

1.3 Block Categories

Blocks are organized into three categories:

Note | Primary Blocks

- `definition` — Define concepts
- `theorem` — State theorems
- `equation` — Named equations

Note | Supporting Blocks

- `note` — Important information
- `notation` — Explain symbols
- `analysis` — Discussion and analysis

Note | Proofs & Examples

- `proof` — Mathematical proofs
- `example` — Worked examples
- `solution` — Solutions (visibility controlled by config)

1.4 Your First Block

Example | Creating a Definition

```
#definition("Velocity")[
  The rate of change of position with respect to time:
  $ v = dif x / dif t $
]
```

Renders as:

Definition | Velocity

The rate of change of position with respect to time:

$$v = \mathrm{d}\frac{x}{\mathrm{d}}t$$

Chapter 01.02

All Block Types

1 All Block Types

A complete reference of every block type in the Block module.

1.1 Primary Blocks

Definition | Definition Block

Use `#definition("Title")[...]` to define concepts.

Theorem | Theorem Block

Use `#theorem("Title")[...]` to state theorems.

Equation | Equation Block

Use `#equation("Title")[...]` for named equations:

$$E = mc^2$$

1.2 Supporting Blocks

Note | Note Block

Use `#note("Title")[...]` for important notes and tips.

Notation | Notation Block

Use `#notation("Title")[...]` to explain mathematical notation and symbols.

Analysis | Analysis Block

Use `#analysis("Title")[...]` for analysis, discussion, and elaboration.

1.3 Proofs and Examples

Proof | Simple Proof

Use `#proof[...]` or `#proof("Title")[...]` for mathematical proofs.

The proof block has a special QED marker at the end.

Example | Example with Solution

Use `#example("Title")[...]` for worked examples.

Solutions can be nested inside examples:

Solution 1 |

Use `#solution[...]` for solutions.

Visibility is controlled by `show-solution` in `config/constants.json`.

1.4 Nesting Blocks

Blocks can be nested for complex content:

Theorem | Fundamental Theorem

A theorem statement here.

Proof |

The proof of the theorem.

Example | Application

An example applying the theorem.

Solution 1 |

The worked solution.

1.5 Styling

Block colors are determined by your active theme. See `config/schemes/` to customize.

Chapter 02

Shape Module

2D geometric primitives: points, lines, circles, polygons.

Chapter 02.01

Points & Lines

1 Points & Lines

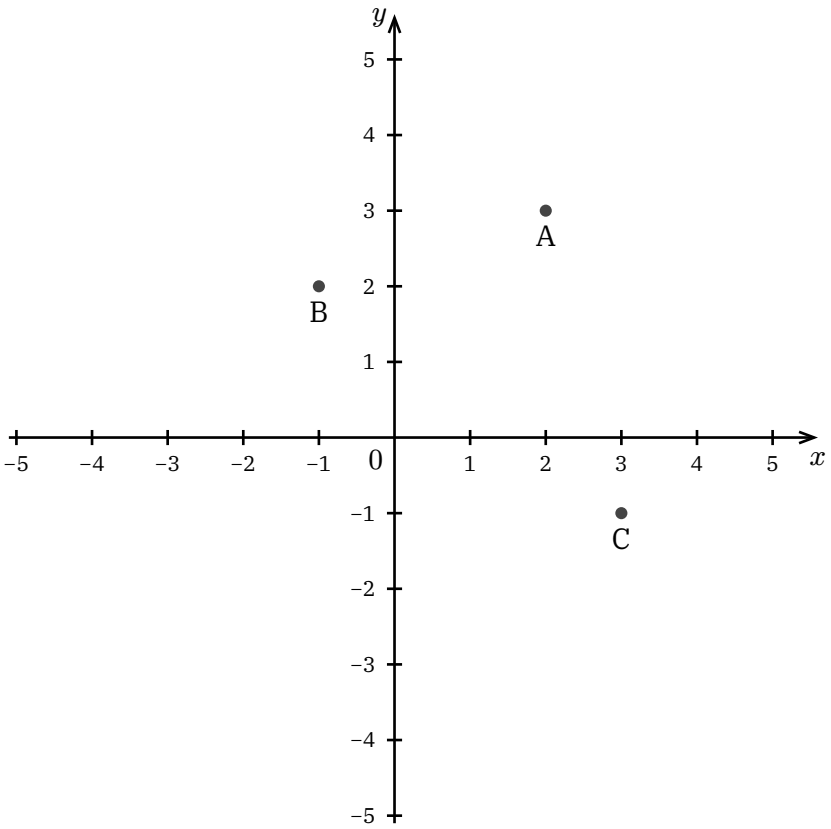
The Shape module provides 2D geometric primitives.

1.1 Creating Points

Definition | point

Creates a point at coordinates (x, y).

```
point(x, y, label: "A", style: auto)
```

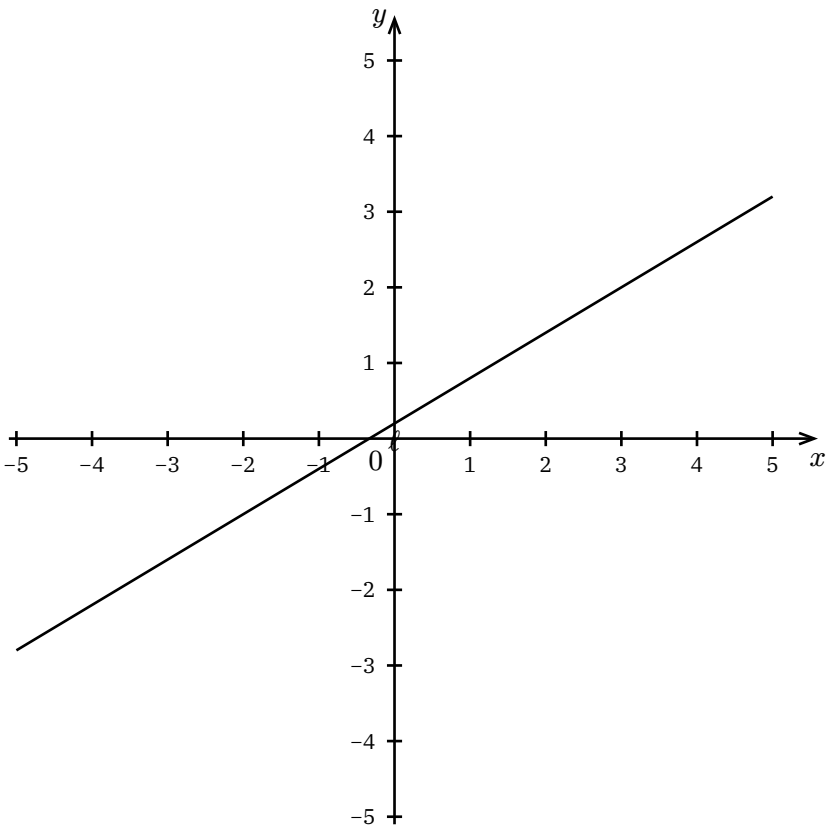


1.2 Creating Lines

Definition | line

Creates an infinite line through two points.

```
line(p1, p2, label: none, style: auto)
```



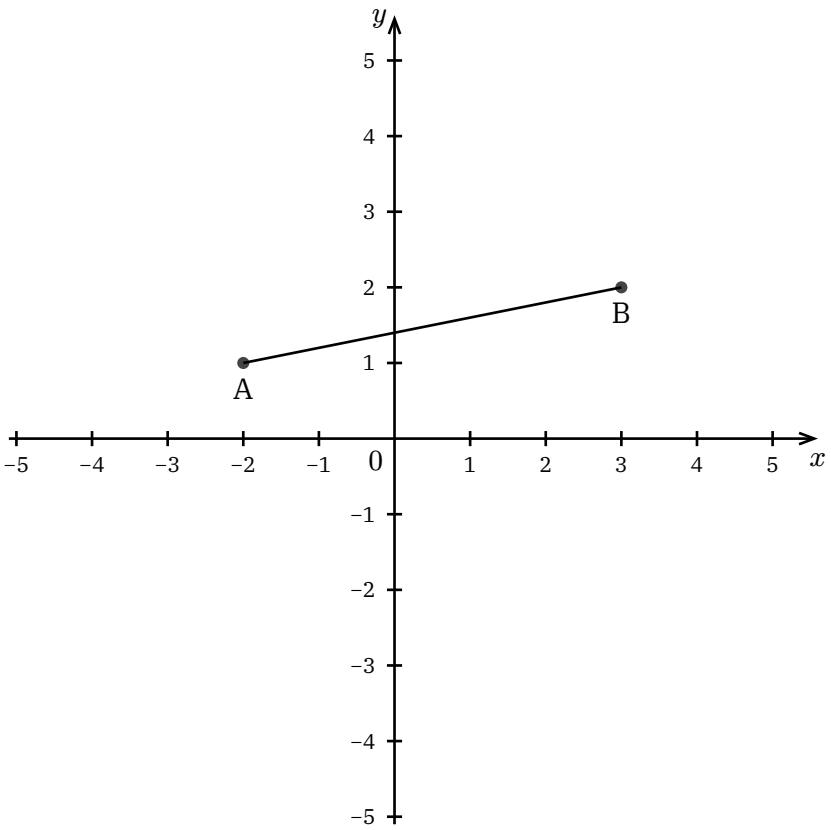
1.3 Line Segments

Use `segment` for lines with definite endpoints:

Definition | segment

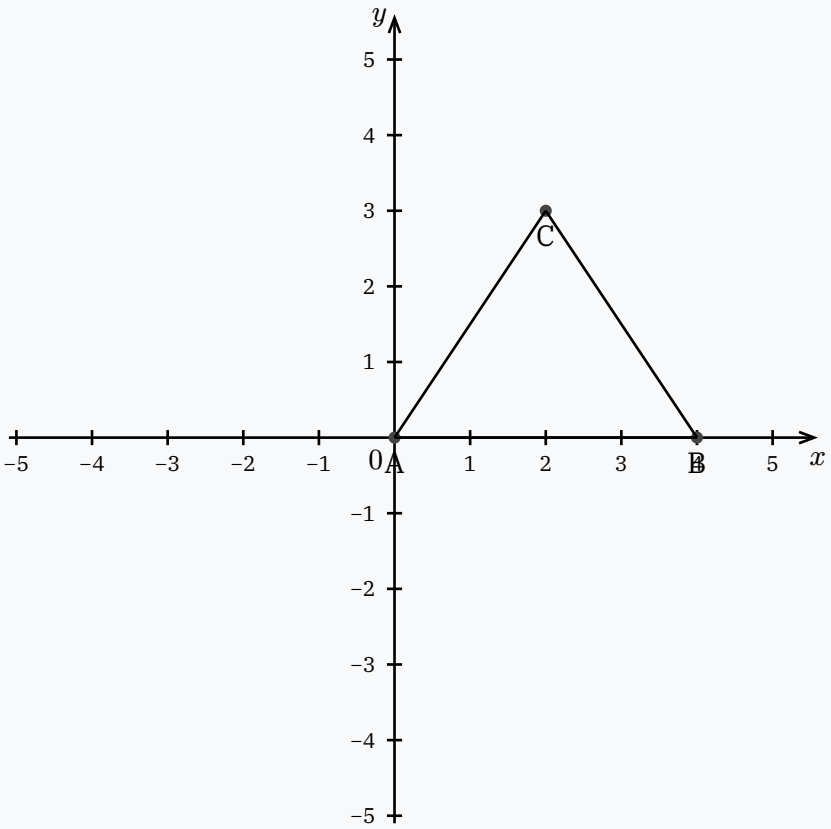
Creates a finite line segment between two points.

```
segment(p1, p2, label: none, style: auto)
```



1.4 Combining Points and Lines

Example | Triangle Vertices


y
x
0
A
B
C
5
4
3
2
1
−1
−2
−3
−4
−5
−5
−4
−3
−2
−1
1
2
3
5

Chapter 02.02

Circles & Polygons

1 Circles & Polygons

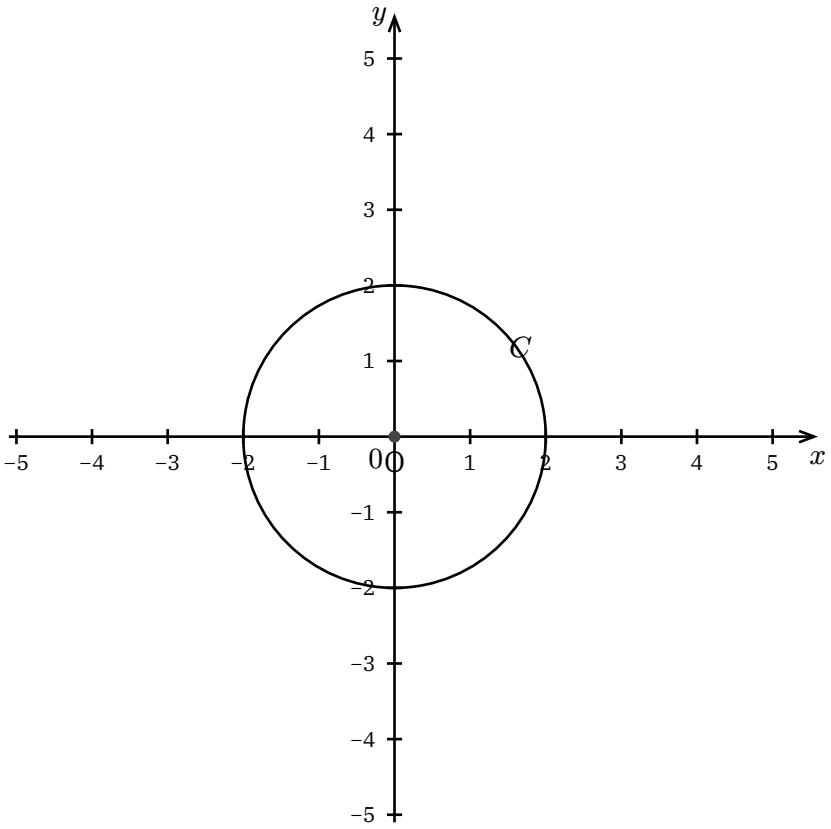
Create circles and multi-sided shapes.

1.1 Circles

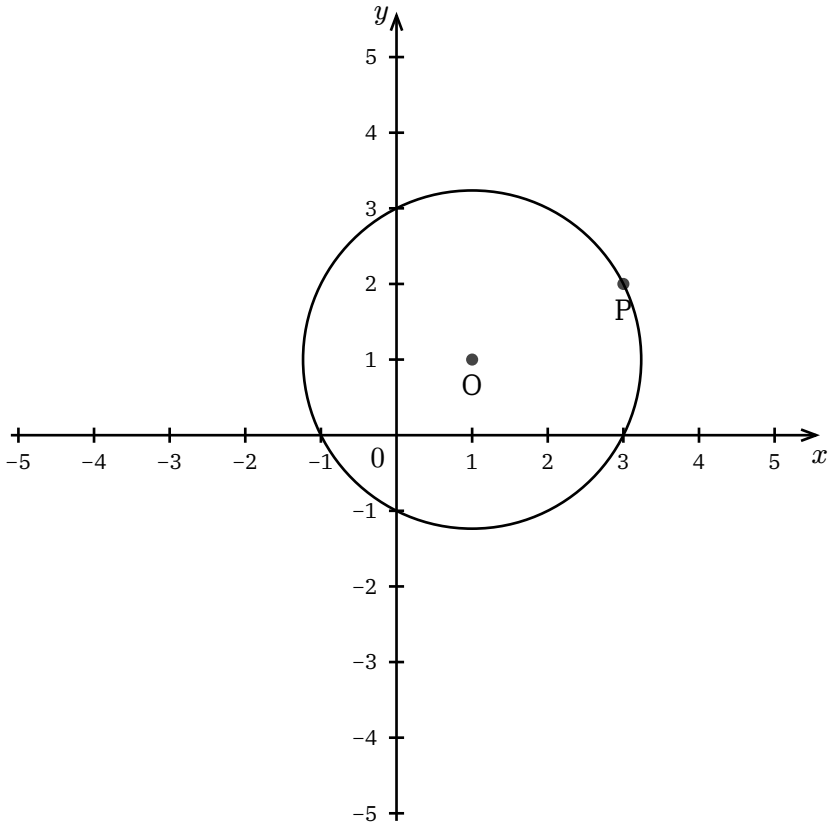
Definition | circle

Creates a circle from center and radius, or center and a point on the circle.

```
circle(center, radius: r, label: none, style: auto)
circle(center, through: point, label: none, style: auto)
```



1.2 Circle Through Point

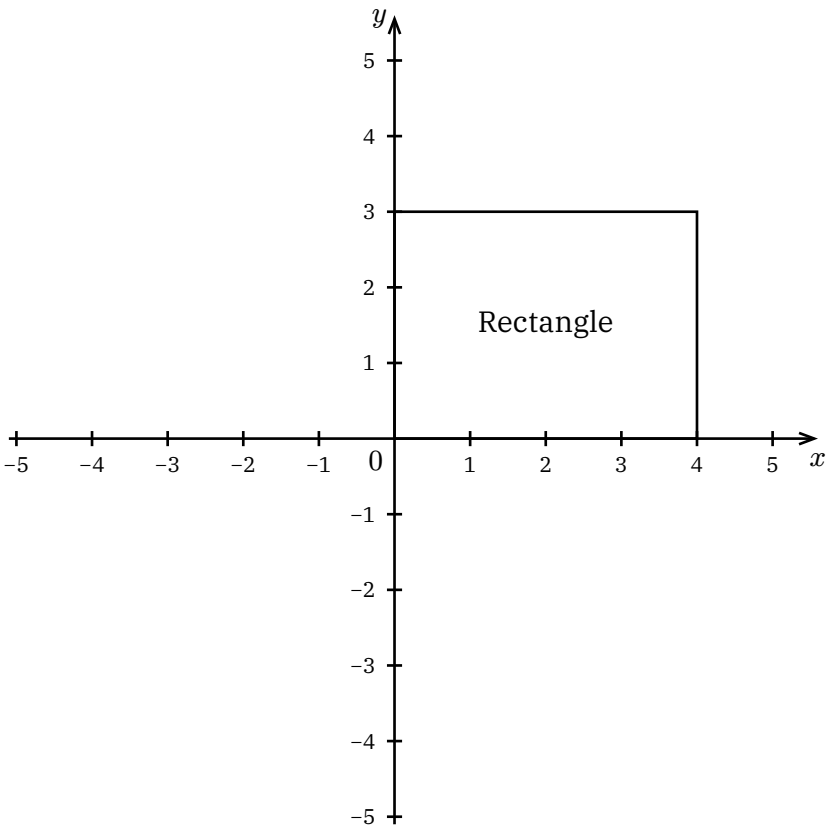


1.3 Polygons

Definition | polygon

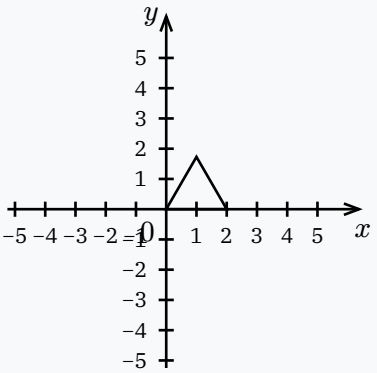
Creates a closed polygon from vertices.

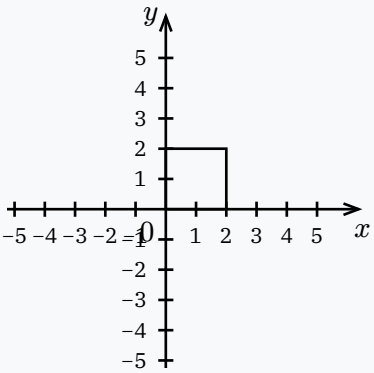
```
polygon(p1, p2, p3, ..., label: none, style: auto)
```



1.4 Regular Polygons

Example | Regular Shapes



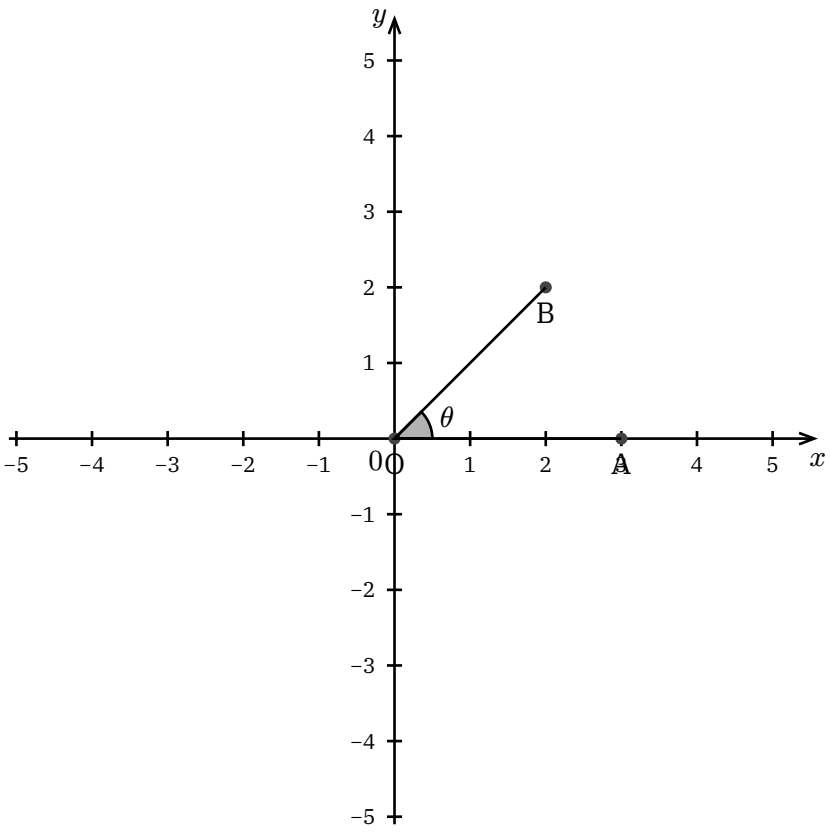


1.5 Angles

Definition | angle

Creates an angle marker between three points.

```
angle(p1, vertex, p2, label: $theta$, style: auto)
```


y
x
O
A
B
θ
5
4
3
2
1
−1
−2
−3
−4
−5
−5
−4
−3
−2
−1
1
2
4
5

Chapter 02.03

Intersections & Constructions

1 Intersections & Constructions

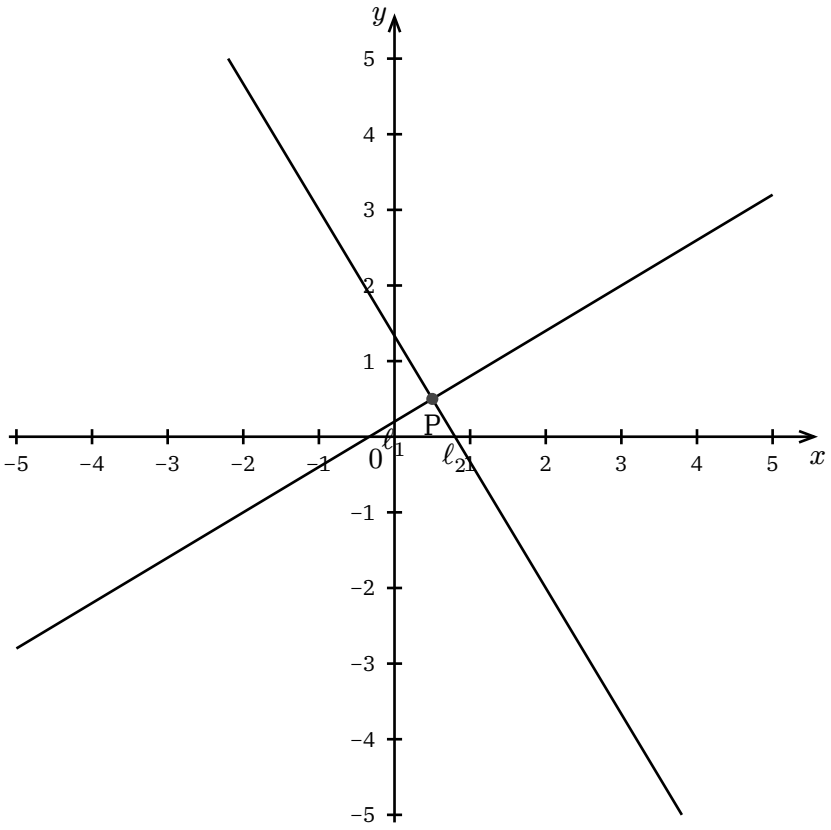
Find intersections and construct derived objects.

1.1 Line-Line Intersection

Definition | intersect-ll

Finds the intersection of two lines.

```
intersect-ll(line1, line2, label: "P")
```

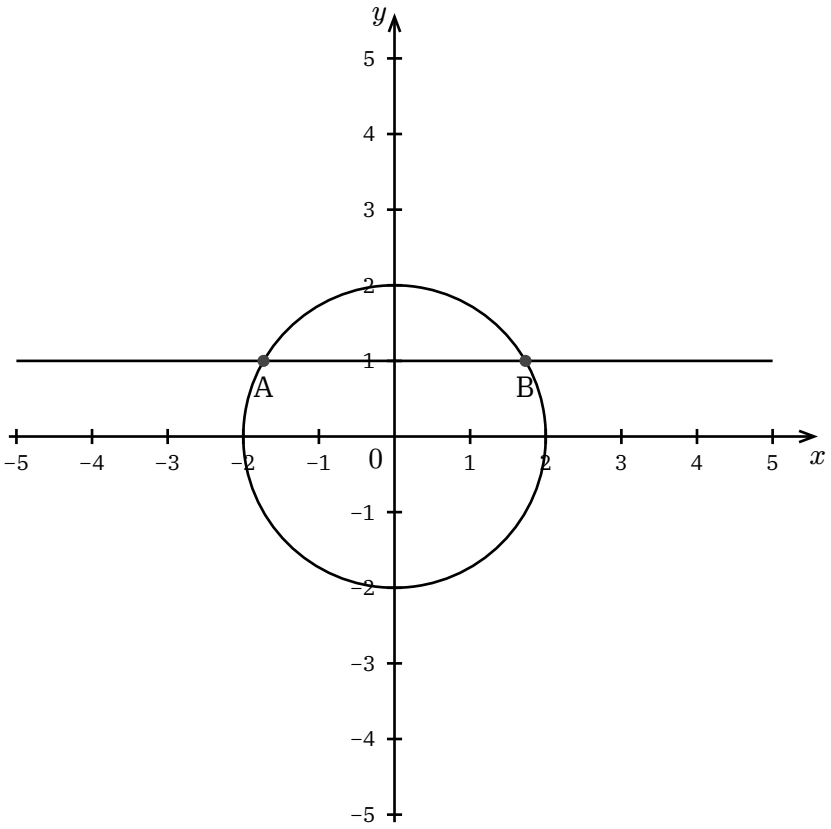


1.2 Line-Circle Intersection

Definition | intersect-lc

Finds intersections of a line and circle.

```
intersect-lc(line, circle, labels: ("A", "B"))
```

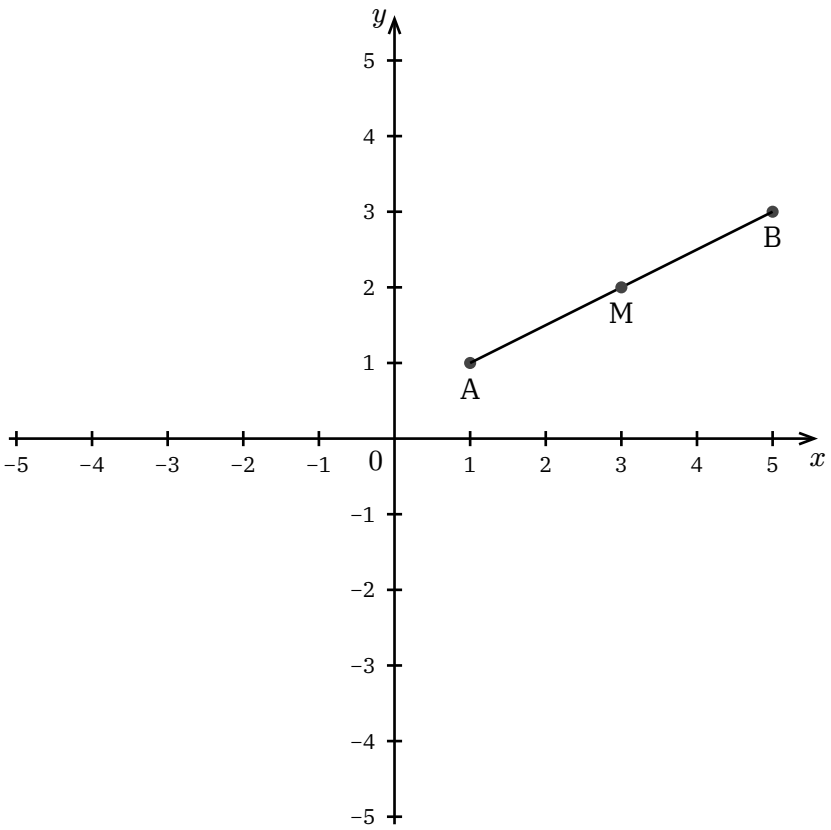


1.3 Constructions

Definition | midpoint

Constructs the midpoint of a segment.

```
midpoint(p1, p2, label: "M")
```



1.4 Perpendicular & Parallel

Definition | perpendicular

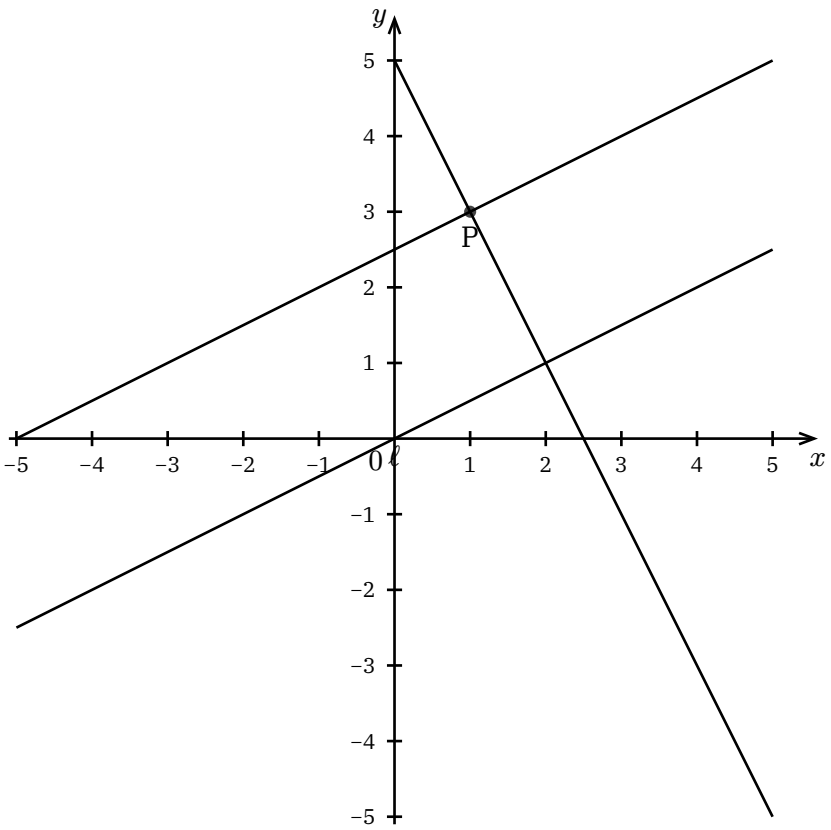
Constructs a line perpendicular to a given line through a point.

```
perpendicular(line, point, label: none)
```

Definition | parallel

Constructs a line parallel to a given line through a point.

```
parallel(line, point, label: none)
```


y
x
P
O
ℓ
5
4
3
2
1
−1
−2
−3
−4
−5
−5
−4
−3
−2
−1
1
2
3
4
5

Chapter 03

Graph Module

Functions, vectors, and calculus operations.

Chapter 03.01

Function Plotting

1 Function Plotting

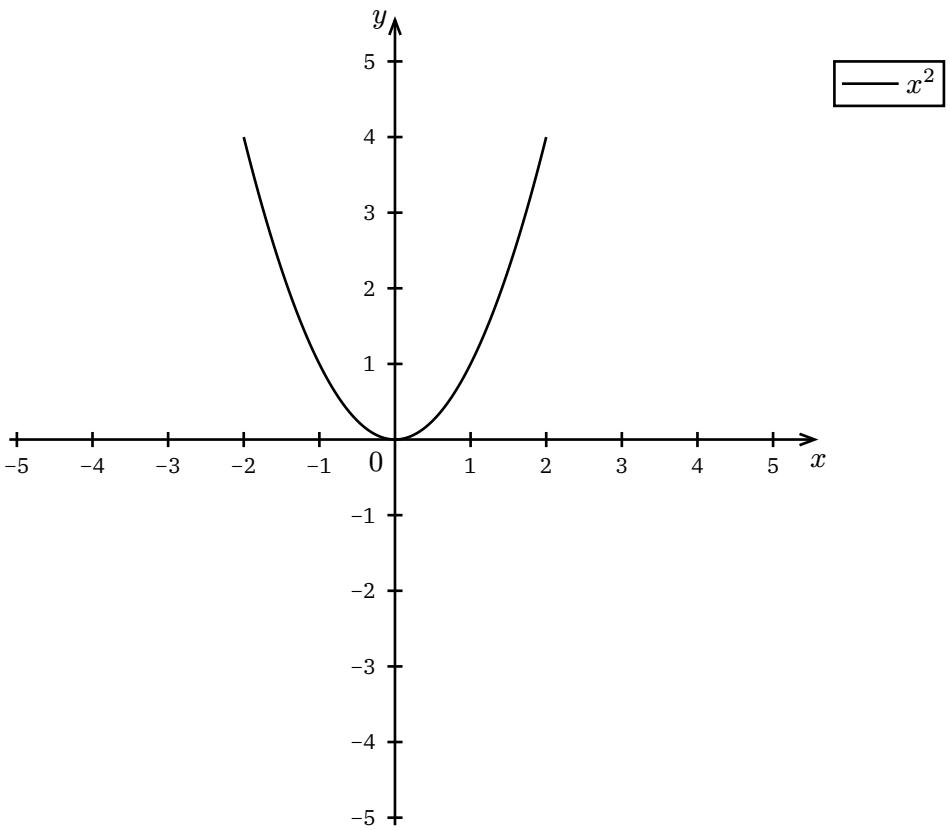
The Graph module provides function plotting and mathematical visualization.

1.1 The graph Function

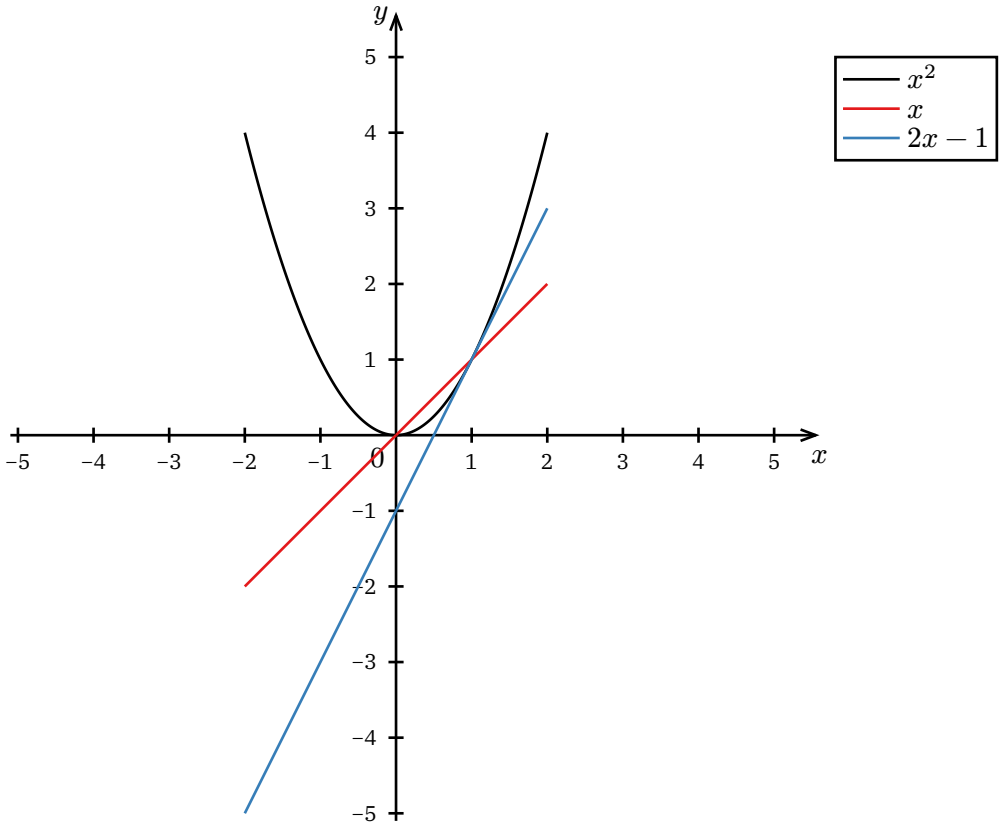
Definition | graph

Plots a function $y = f(x)$ over a domain.

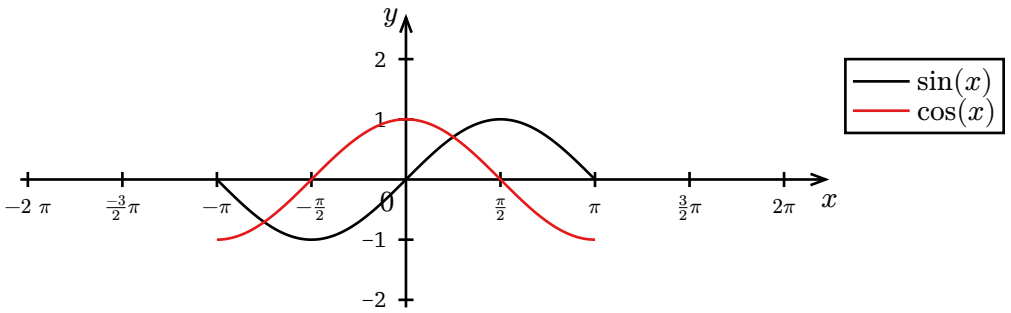
```
graph(x => expr, domain: (min, max), label: $f(x)$)
```



1.2 Multiple Functions



1.3 Trigonometric Functions

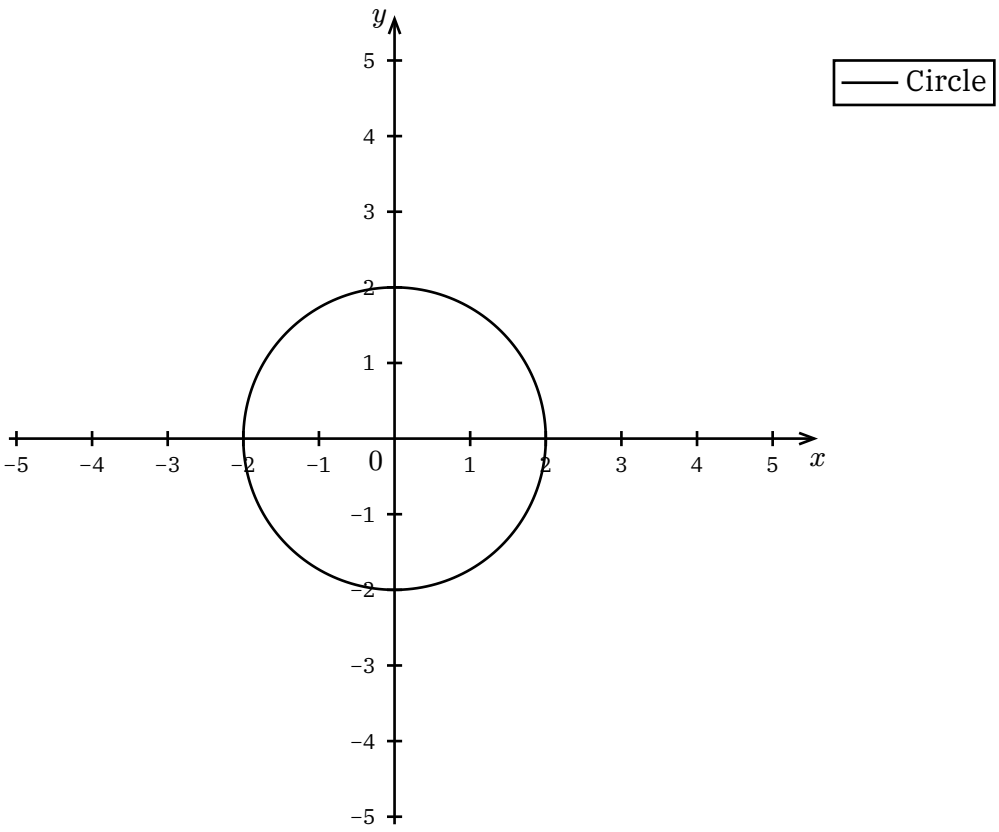


1.4 Parametric Functions

Definition | parametric

Plots a parametric curve $(x(t), y(t))$.

```
parametric(t => (x(t), y(t)), domain: (min, max), label: none)
```


y
x
0
Circle
5
4
3
2
1
−1
−2
−3
−4
−5
−5
−4
−3
−2
−1
1
2
3
4
5

Chapter 03.02

Vectors

1 Vectors

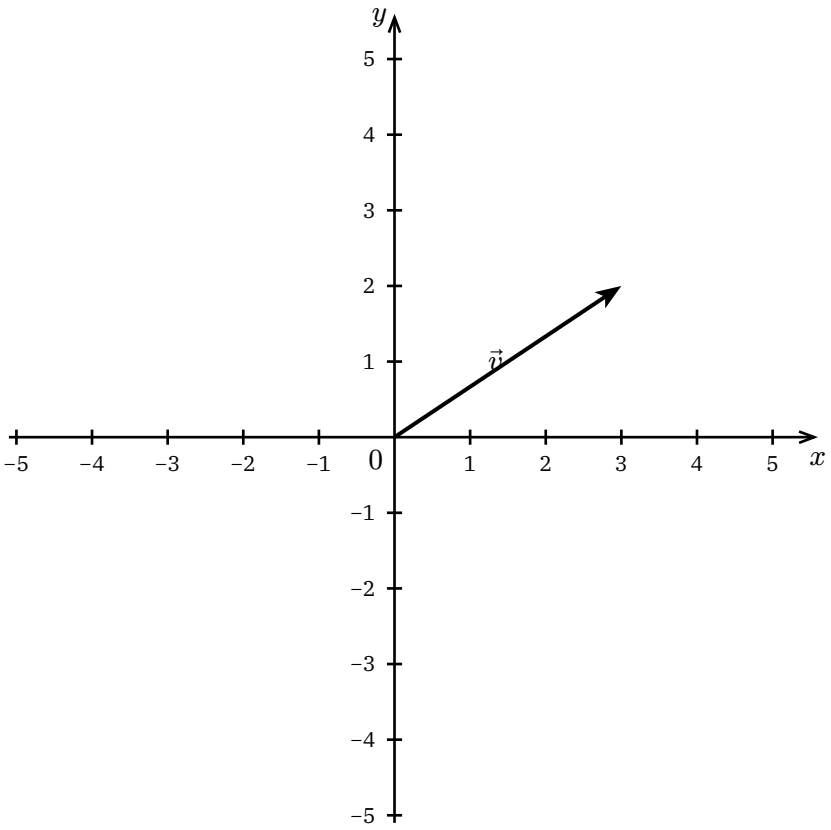
The Graph module includes vector operations for 2D vector mathematics.

1.1 Creating Vectors

Definition | vec

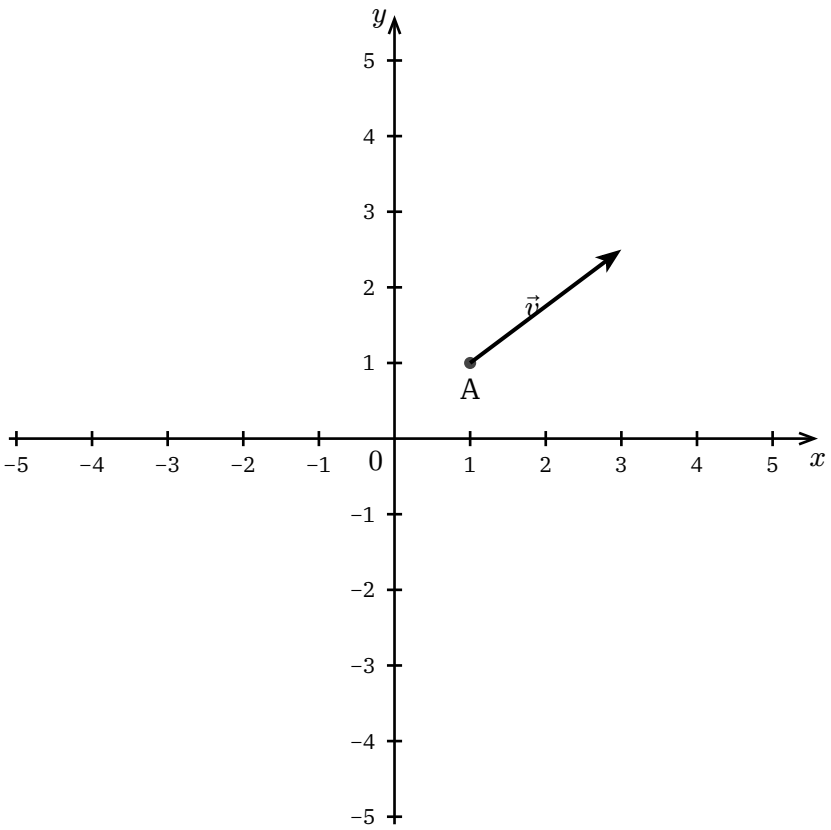
Creates a 2D vector object.

```
vec((x, y), label: $arrow(v)$, origin: (0, 0))
```



1.2 Vector from Point

Vectors can start from any origin:

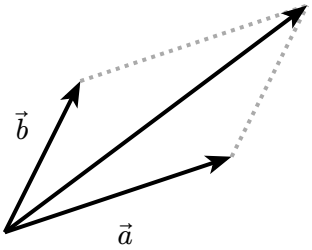


1.3 Vector Addition

Definition | vec-add

Visualizes vector addition with parallelogram.

```
vec-add(v1, v2, helplines: true)
```

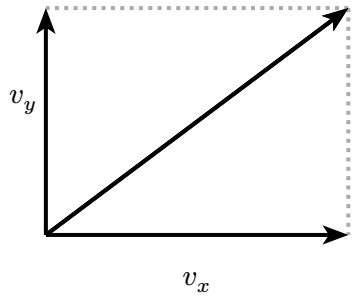


1.4 Vector Components

Definition | vec-components

Shows vector decomposition into components.

```
vec-components(v, labels: ($v_x$, $v_y$))
```

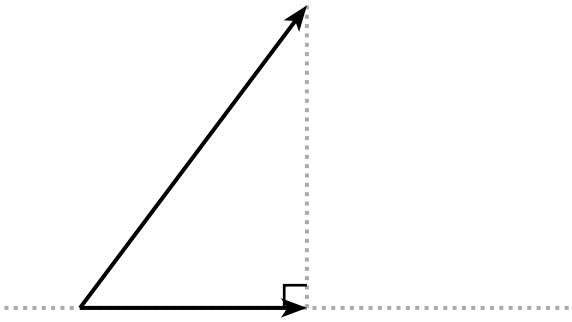


1.5 Vector Projection

Definition | vec-project

Projects one vector onto another.

```
vec-project(v, onto: w, helplines: true)
```



Chapter 04

Canvas Module

Plotting canvases for rendering shapes and graphs.

Chapter 04.01

Cartesian Canvas

1 Cartesian Canvas

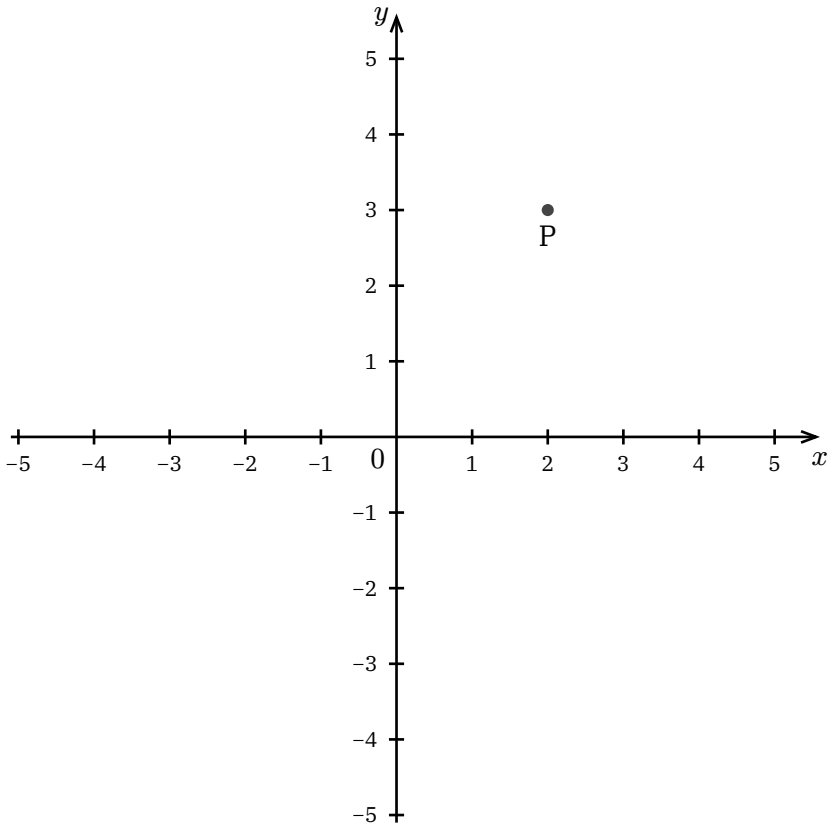
The Canvas module provides rendering surfaces for shapes and graphs.

1.1 Basic Canvas

Definition | cartesian-canvas

Creates a 2D Cartesian coordinate system.

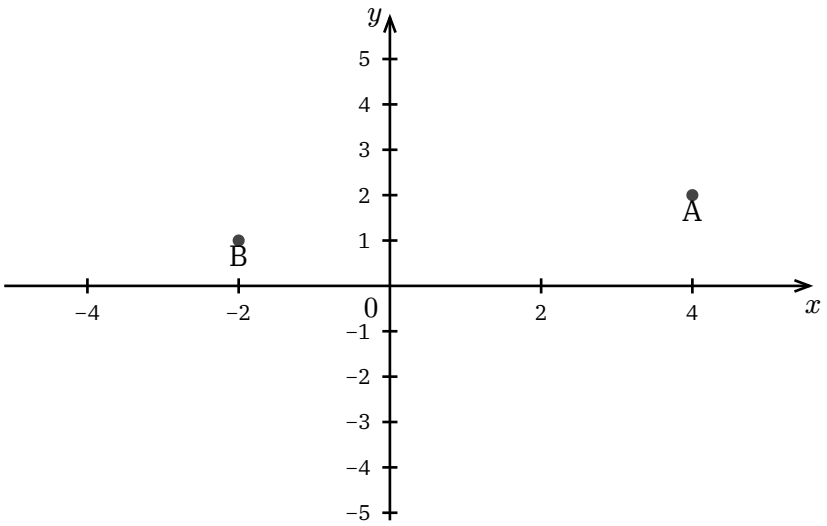
```
cartesian-canvas(
  width: 8cm, height: 6cm,
  x-tick: 1, y-tick: 1,
  ..objects
)
```



1.2 Canvas Options

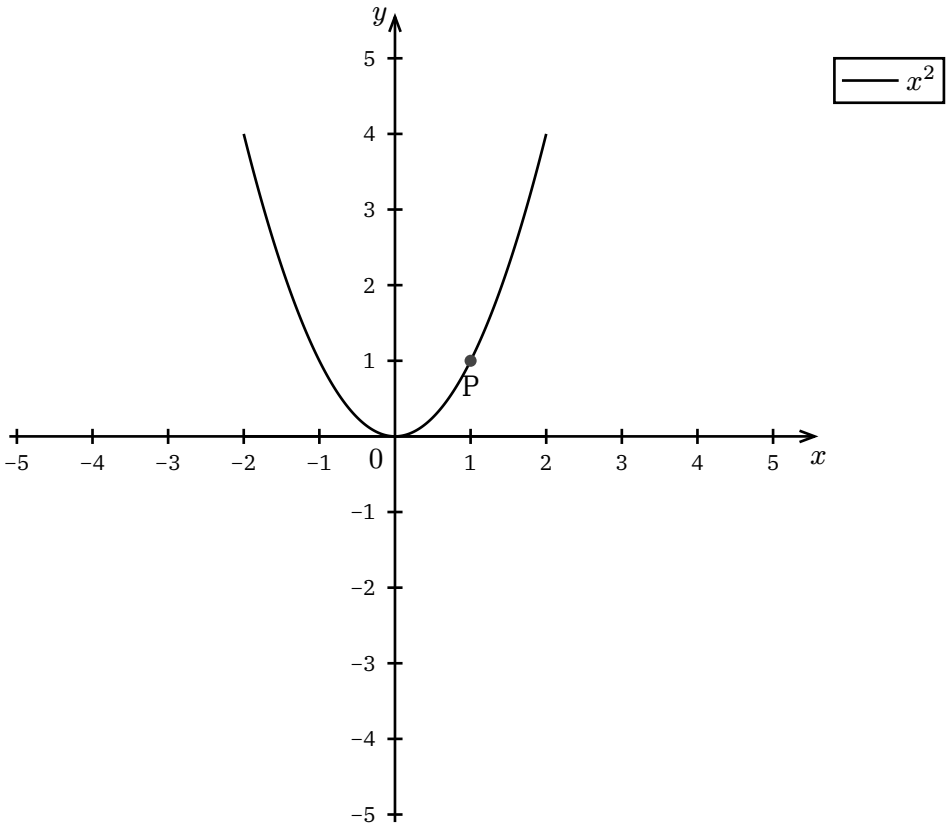
Notation | Key Parameters

- `width`, `height` – Canvas dimensions
- `x-tick`, `y-tick` – Grid spacing
- `x-label`, `y-label` – Axis labels
- `show-grid` – Toggle grid visibility



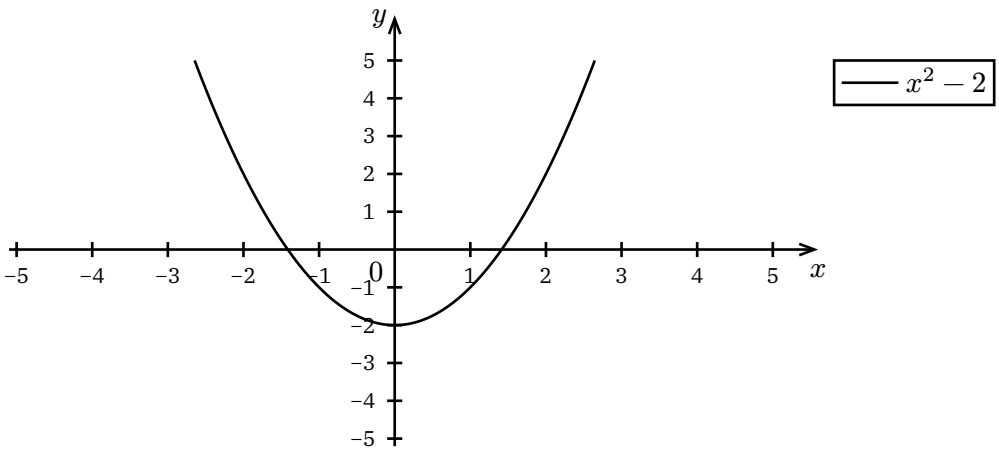
1.3 Combining Shapes and Graphs

The cartesian canvas can display both shapes and graphs:



1.4 Graph Canvas

For simpler function-only plots, use `graph-canvas`:


y
x
0
5
4
3
2
1
−1
−2
−3
−4
−5
−5
−4
−3
−2
−1
1
2
3
4
5
$x^2 - 2$

Chapter 04.02

Polar & Trig Canvas

1 Polar & Trig Canvas

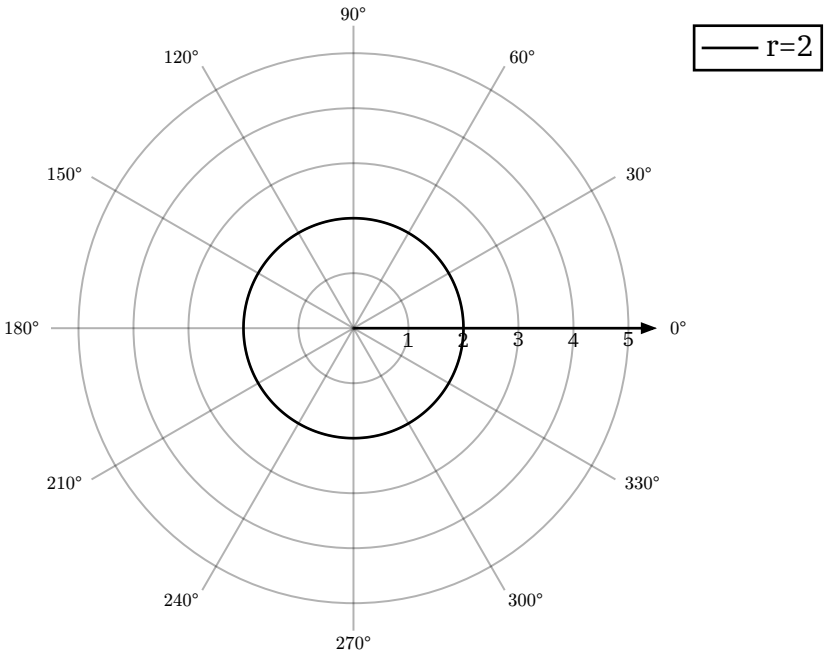
Specialized canvases for polar coordinates and trigonometry.

1.1 Polar Canvas

Definition | polar-canvas

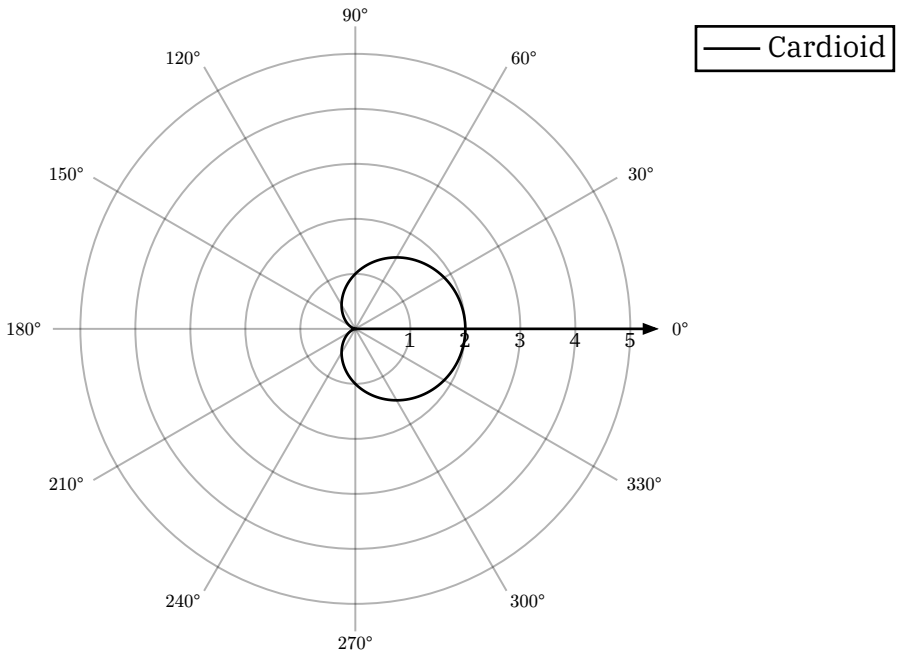
Creates a polar coordinate system with radial and angular axes.

```
polar-canvas(
  width: 8cm,
  r-max: 3,
  ..objects
)
```



1.2 Polar Functions

Use `polar-func` to plot $r = f(\theta)$:

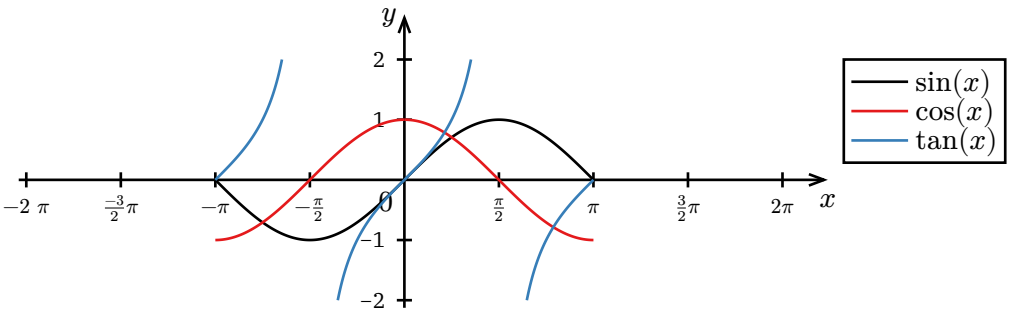


1.3 Trig Canvas

Definition | trig-canvas

A Cartesian canvas with ticks at multiples of pi.

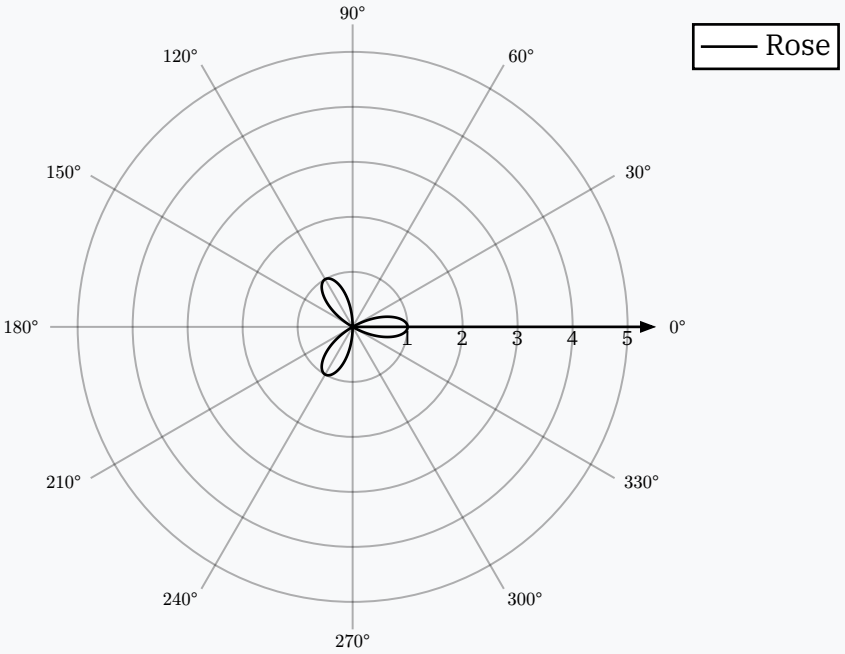
```
trig-canvas(
  width: 10cm,
  ..objects
)
```



1.4 Rose Curves

Example | Polar Rose

$r = \cos(3\theta)$ creates a 3-petal rose:


Rose
90°
120°
60°
150°
30°
180°
0°
1
2
3
4
5
210°
330°
240°
300°
270°

Chapter 04.03

3D Space Canvas

1 3D Space Canvas

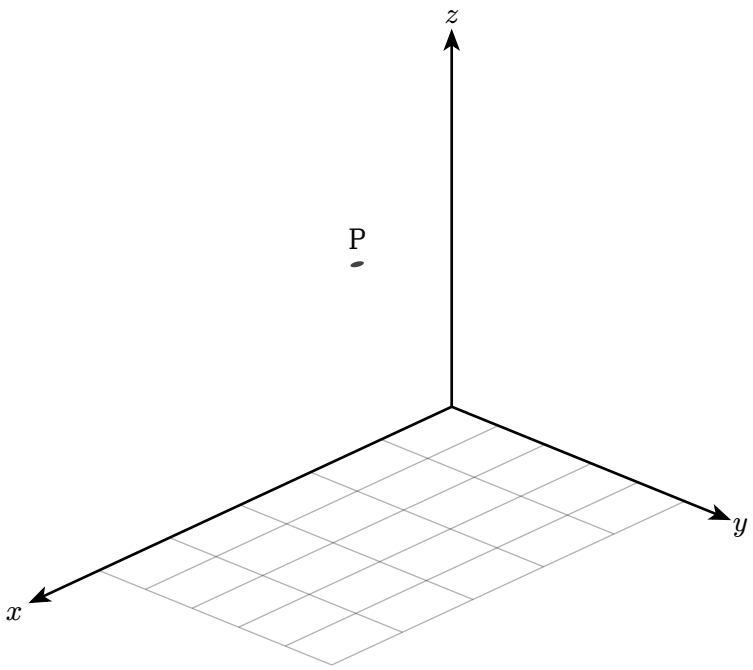
Visualize 3D geometry and vectors.

1.1 Space Canvas

Definition | space-canvas

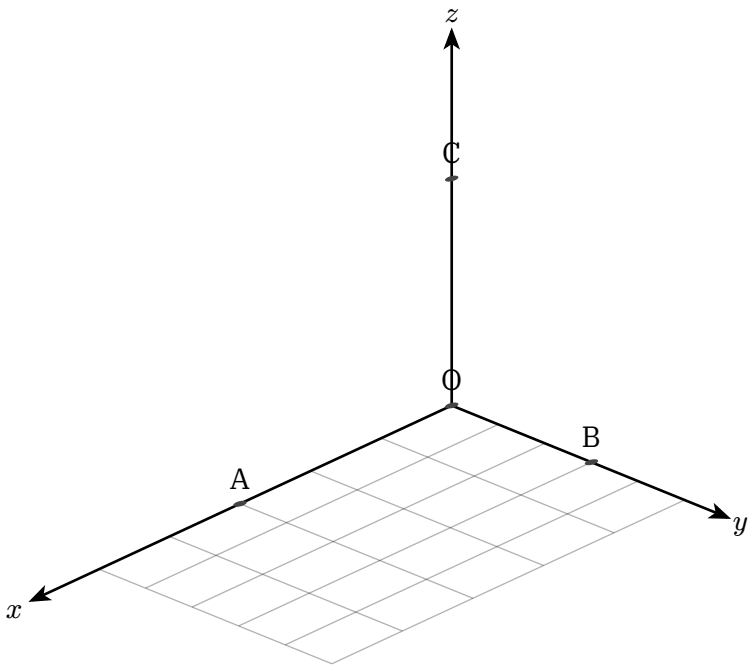
Creates a 3D coordinate system with perspective.

```
space-canvas(
  width: 8cm,
  ..objects
)
```



1.2 3D Points

Use `point()` with z coordinate for 3D points:



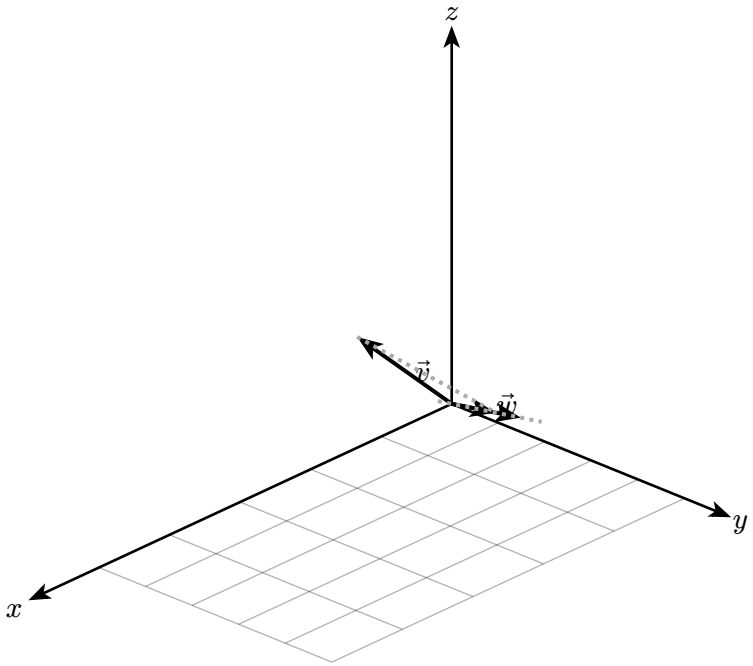
1.3 3D Vectors

Use `vec()` with 3 components for 3D vectors:

Definition | vec (3D)

Creates a 3D vector from origin.

```
vec((x, y, z), label: $arrow(v)$)
```



1.4 Coordinate Axes

The space canvas follows the right-hand rule:

- x-axis points right
- y-axis points forward
- z-axis points up

Chapter 05

Data Module

Data visualization: tables, series, and curves.

Chapter 05.01
Tables

1 Tables

The Data module provides table rendering with theme-aware styling.

1.1 Table Plot

Definition | table-plot

Creates a styled data table.

```
table-plot(
  headers: ("x", "y", "z"),
  data: ((1, 2, 3), (4, 5, 6)),
)
```

Variable	Mean	Std Dev
Height	175 cm	8.5
Weight	70 kg	12.3
Age	25 yr	4.2

1.2 Value Table

Definition | value-table

Creates a function value table with variable and result rows.

```
value-table(
  variable: $x$,
  func: $f(x)$,
  values: (1, 2, 3, 4),
  results: (1, 4, 9, 16),
)
```

x	x^2
−2	4
−1	1

0	0
1	1
2	4

1.3 Grid Table

Definition | grid-table

Creates a grid layout for 2D data visualization.

```
grid-table(
  data: ((1, 2, 3), (4, 5, 6)),
  show-indices: true,
)
```

0	1	2
1	2	3
4	5	6
7	8	9

1.4 Compact Table

For inline or small tables:

n	n!
0	1
1	1
2	2
3	6
4	24

Chapter 05.02

Data Series & CSV

1 Data Series & CSV

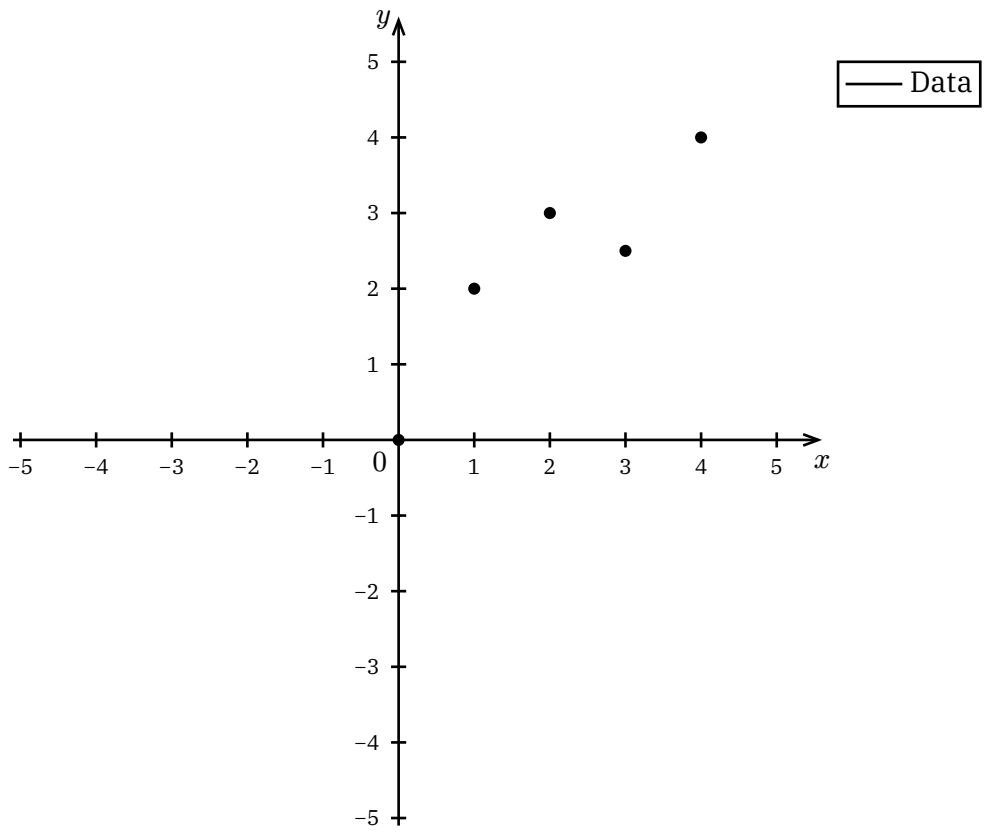
Plot data points from arrays or CSV files.

1.1 Data Series

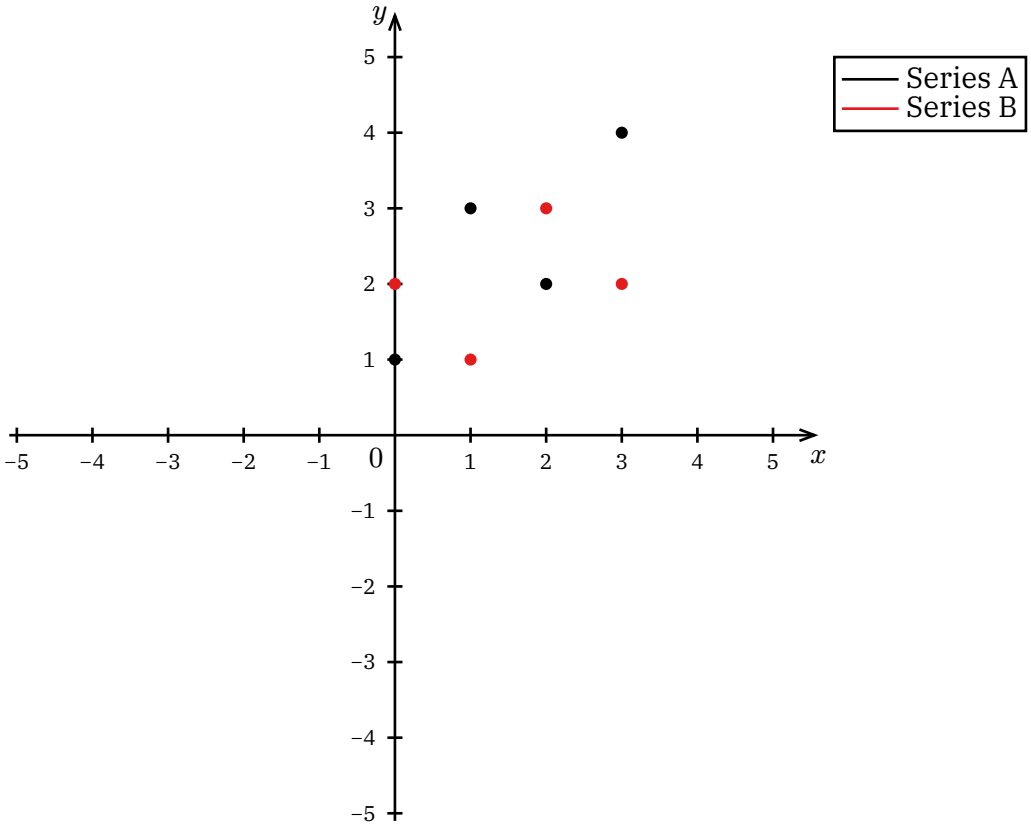
Definition | data-series

Creates a plotable data series from coordinate pairs.

```
data-series(
  ((x1, y1), (x2, y2), ...),
  label: "Series",
  style: auto,
)
```



1.2 Multiple Series



1.3 CSV Import

Definition | csv-series

Loads data from a CSV file.

```
csv-series(
  "path/to/data.csv",
  x-col: 0,
  y-col: 1,
  label: "CSV Data",
)
```

Note | CSV Format

The CSV file should have numeric data. Header rows are automatically detected and skipped.

1.4 Polar Data Series

Definition | polar-data-series

Creates a data series in polar coordinates (r, θ).

```
polar-data-series(
  ((r1, θ1), (r2, θ2), ...),
  label: "Polar",
)
```

Use `polar-data-series` with `polar-canvas` for radial data visualization.

Chapter 05.03

Smooth Curves

1 Smooth Curves

Draw smooth curves through data points using spline interpolation.

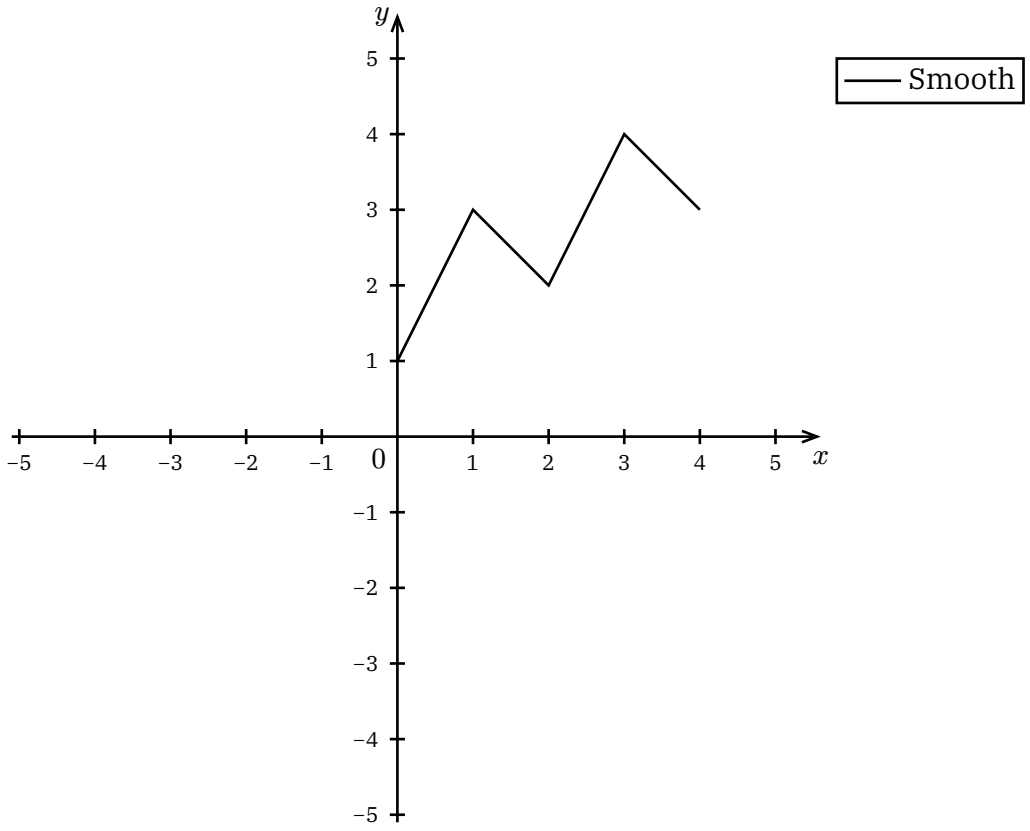
1.1 Curve Through Points

Definition | curve-through

Creates a smooth curve through a set of points.

```
curve-through(
  (p1, p2, p3, ...),
  label: "Curve",
  tension: 0.5,
)
```

- `tension`: Controls curve tightness (0 = linear, 1 = tight)

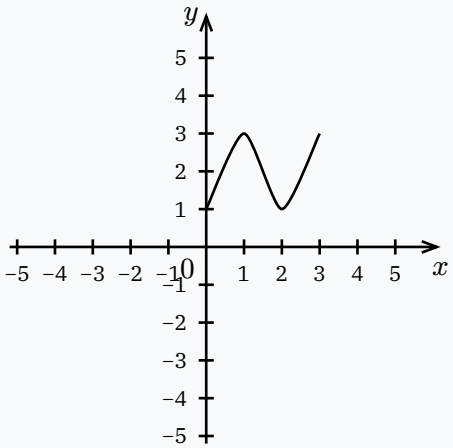


1.2 Tension Control

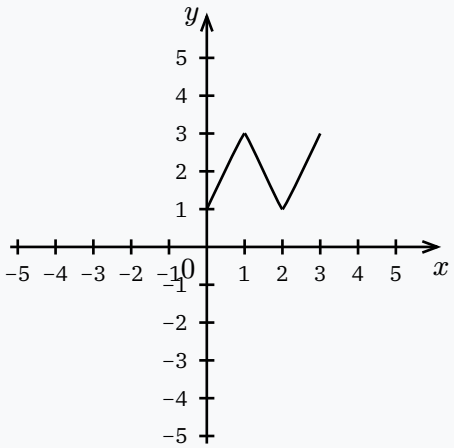
Example | Tension Comparison

Lower tension creates smoother curves:

Tension: 0.3



Tension: 0.8

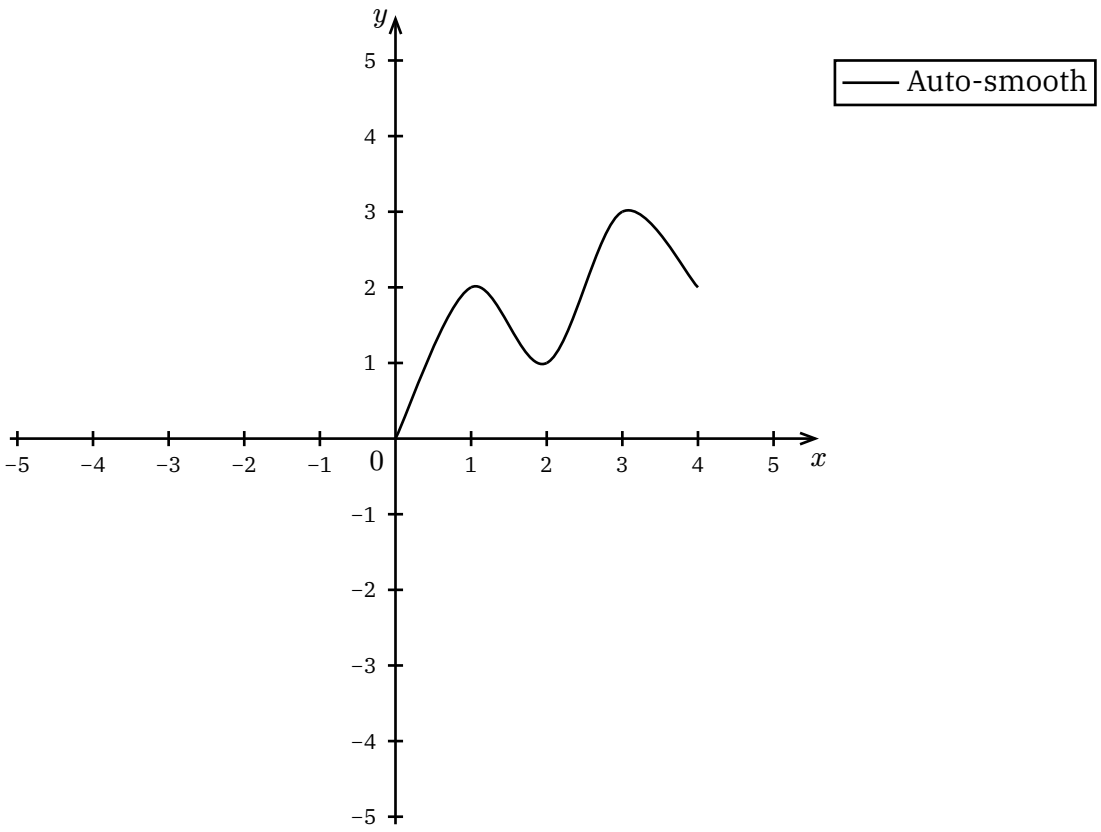


1.3 Smooth Curve

Definition | smooth-curve

Alternative curve function with automatic tension.

```
smooth-curve(
  (p1, p2, p3, ...),
  label: "Curve",
)
```


y
x
0
5
4
3
2
1
−1
−2
−3
−4
−5
−5
−4
−3
−2
−1
1
2
3
4
5
Auto-smooth

Chapter 06

Cover Module

Document covers and title pages.

Chapter 06.01

Cover Templates

1 Cover Templates

The Cover module provides document covers and title pages.

1.1 Main Cover

Definition | cover

The main document cover, shown at the beginning. Configured automatically via `config/metadata.json`.

Note | Configuration

Cover content is set in `config/metadata.json`:

```
{
  "title": "Your Document Title",
  "subtitle": "Optional Subtitle",
  "authors": ["Author 1", "Author 2"],
  "affiliation": "Your Institution"
}
```

1.2 Chapter Cover

Definition | chapter-cover

Shown at the start of each chapter. Configured via `hierarchy.json`:

```
{
  "title": "Chapter Title",
  "summary": "Brief chapter description."
}
```

Controlled by `display-chap-cover` in `constants.json`.

1.3 Preface

Definition | preface

Introduction page shown after the cover. Content is in `config/preface.typ`.

Edit `config/preface.typ` to add your preface content.

1.4 Project (Page Title)

Definition | project

Individual page headers. Each page displays its title from `hierarchy.json`.

1.5 Display Controls

In `config/constants.json`:

Notation | Display Flags

- `display-cover` — Show main cover
- `display-outline` — Show table of contents
- `display-chap-cover` — Show chapter covers
- `display-mode` — Theme name (e.g., “noteworthy-dark”)

Chapter 07

Layout Module

Page layouts, outlines, and configuration.

Chapter 07.01

Layout & Config

1 Layout & Config

The Layout module handles table of contents and page structure. Configuration options control project-wide settings.

1.1 Table of Contents

Definition | outline

Automatically generated table of contents based on your `hierarchy.json`.

The outline displays:
- Chapter numbers and titles
- Page numbers and titles
- Correct page numbering

Controlled by `display-outline` in `constants.json`.

1.2 Heading Numbering

Notation | Numbering Format

Configure in `constants.json`:

```
{
  "heading-numbering": "1.1",
  "pad-chapter-id": true,
  "pad-page-id": true
}
```

- `heading-numbering` — Format for section headings
- `pad-chapter-id` — Zero-pad chapter numbers (01, 02...)
- `pad-page-id` — Zero-pad page numbers (01.01, 01.02...)

1.3 Solutions Visibility

Note | Show/Hide Solutions

Control solution block visibility:

```
{
  "show-solution": true,
```

```
    "solutions-text": "Solutions",
    "problems-text": "Problems"
  }
```

When `show-solution` is `false`, all `#solution[...]` blocks are hidden.

1.4 Font Configuration

```
{
  "font": "Linux Libertine",
  "title-font": "Inter"
}
```

1.5 Building Your Document

Use the Noteworthy TUI:

```
python3 noteworthy.py
```

Select **Builder** → Choose chapters → Press **Enter** to build.

The output PDF is saved to `output.pdf`.

Chapter 08

Combi Module

Combinatorics visualizations: permutations, combinations, and counting.

Chapter 08.01

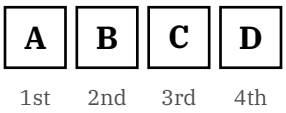
Combinatorics Visualizations

1 Combinatorics Visualizations

Visual representations for counting problems.

1.1 Linear Permutations

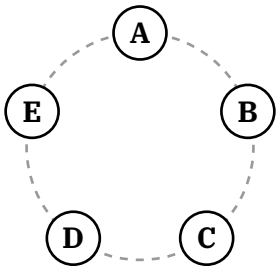
Arrange items in a row:



Highlight specific positions:

1.2 Circular Permutations

Arrange items in a circle:



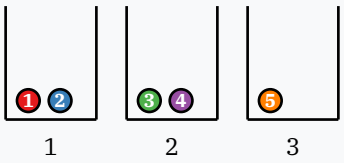
1.3 Balls and Boxes

Distribute balls into boxes:

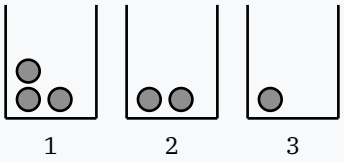
Definition | balls-boxes

Visualize distribution problems:
- Distinguishable balls: numbered, colored differently
- Identical balls: same color

Example | Distinguishable Balls

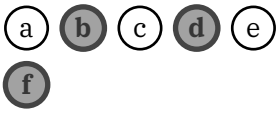


Example | Identical Balls



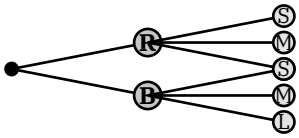
1.4 Subset Selection (Combinations)

Highlight a subset of elements:



1.5 Counting Trees

Visualize multiplication principle:



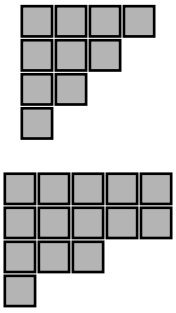
1.6 Partition Diagrams

Ferrers/Young diagram for partitions:

Definition | partition-vis

Shows a partition of n as a Ferrers diagram.

```
partition-vis((4, 3, 2, 1))  // 4 + 3 + 2 + 1 = 10
```



1.7 Pigeonhole Principle

Visualize when items must share containers: