Introduction to Writing with LaTeX

Intermediate Level

Aslak Johansen asjo@mmmi.sdu.dk

October 17, 2025

Part 1: Macros

Introduction

Q: What is a macro?

A: It is a bit like a function, but the evaluation of the body is done by replacing all occurrences of the parameters with the concrete arguments. The resulting text is then inserted in the macro calls place.

Consequence: It can be tricky to think about nested macros, and worse to reason about errors in them.

As the old adage goes, if your macro definitions represents your peak brilliance, then you are mentally incapable of debugging them.

Keyword Markup

Pattern: I want to place certain words in the index, and make it clear that they are special.

Definition:

```
\newcommand{\idx}[2]{\index{#2}\textcolor{purple}{#1}}
```

Use:

```
We need to fit this to the \idx{tradeoff}{Tradeoff}.
```

Appearance:
We need to fit this to the tradeoff.

(and an entry in the index for "Tradeoff")

Highlights

Pattern: I need to refer to something of a particular type (e.g., a file, variable or type), and want it to stand out.

Definition:

```
\newcommand{\filename}[1]{ \textcolor{purple}{\texttt{#1}} }
```

Use:

```
The configuration is pulled from the \filename{conf.yaml} file.
```

Appearance:
The configuration is pulled from the `conf.yaml` file.

Description Lists

Pattern: List items often need a header that stands out over the rest of the item text.

Definition:

```
\newcommand{\textdesc}[1]{ \textit{\textbf{#1}} }
\newcommand{\descitem}[1]{ \item \textdesc{#1} }
```

Use:

```
\begin{itemize}
  \descitem{Bonobo} An intelligent ape.
  \descitem{Gibbon} An agile ape that is less intelligent than the
  ↪  \textdesc{Bonobo}.
\end{itemize}
```

Appearance:

- ***Bonobo*** An intelligent ape.
- ***Gibbon*** An agile ape that is less intelligent than the ***Bonobo***.

Inspirational Quotes [1/2]

Pattern: I want to sprinkle inspirational quotes around my document.

Definition:

```latex
\newenvironment{inspiration}[2][0.9]
{
  \begin{center}
  \newcommand{\saveme}{#2}
  \begin{minipagewithmarginpars}{#1\textwidth}
}
{

  \raggedleft{--- \textsl{\saveme}}
  \end{minipagewithmarginpars}
  \end{center}
}
```

Inspirational Quotes [2/2]

Use:

```
\begin{inspiration}{Jane Goodall}
  You cannot get through a single day without having an impact on the
  world around you. What you do makes a difference, and you have to
  decide what kind of difference you want to make.
\end{inspiration}
```

Appearance:

> You cannot get through a single day without having an impact on the world around you. What you do makes a difference, and you have to decide what kind of difference you want to make.
> – Jane Goodall

Typed Boxes [1/2]

Pattern: I want certain text to be placed in special boxes (e.g., info or conclusion).

Definition:

```
\newenvironment{tbox}[2][0.9]
{
  \begin{center}
    \begin{tabular}{|p{#1\textwidth}|}
      \hline
      \cellcolor[gray]{0.9}
      \textbf{#2} \\
      \hline
      \cellcolor[gray]{0.95}
}{
      \\
      \hline
    \end{tabular}
  \end{center}
}
```

Typed Boxes [2/2]

Use:

```
\begin{tbox}{Intuition}
  The dispatch mechanism is highly concurrent. It might not
  bottleneck the system.
\end{tbox}
```

Appearance:

Intuition
The dispatch mechanism is highly concurrent. It might not bottleneck the system.

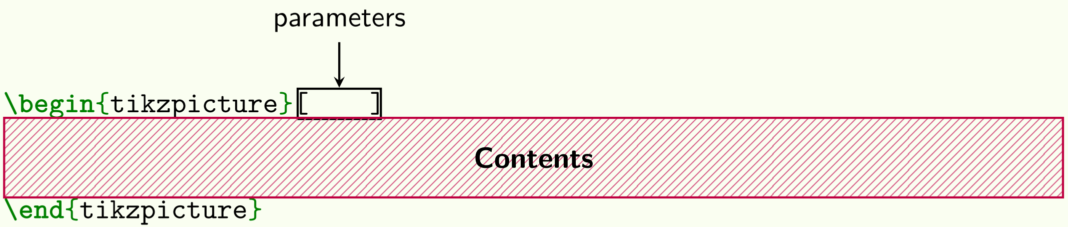
Part 2:
Ti*k*Z

The tikzpicture Environment

Load Ti*k*Z package:

```
\usepackage{tikz}
```

Then we get access to the tikzpicture environment:



```
\begin{tikzpicture}[    ]
\end{tikzpicture}
```

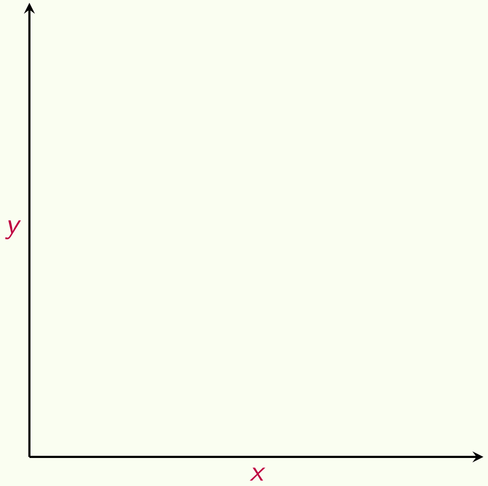
Parameters

The “`remember picture`” parameter allows the `tikzpicture` environment to reference the absolute position of specific points (like nodes or coordinates) on the page.

The “`overlay`” parameter makes LaTeX treat the resulting `tikzpicture` environment as having zero size, thus essentially bypassing the layout system.

These may be combined with a comma in between.

Coordinate System



Nodes

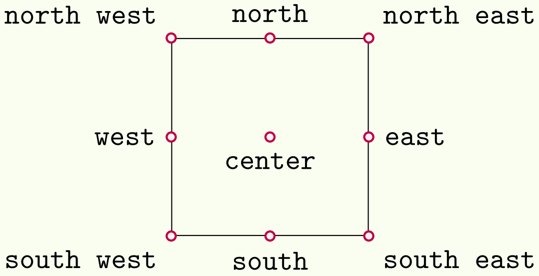
A *node* is something that you can place at a specific position.

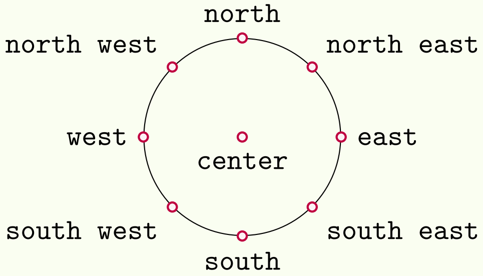
That something is typically some text, but images and math is also allowed.

Properties:

- A shape can be drawn around the node: circle, rectangle, diamond ...
- This material is placed at the *anchor point* of the nodes shape.
- The inside of that shape can be filled with a color or a pattern.
- All of this is *drawn* according to a *style*.

Nodes ▷ Anchor Points

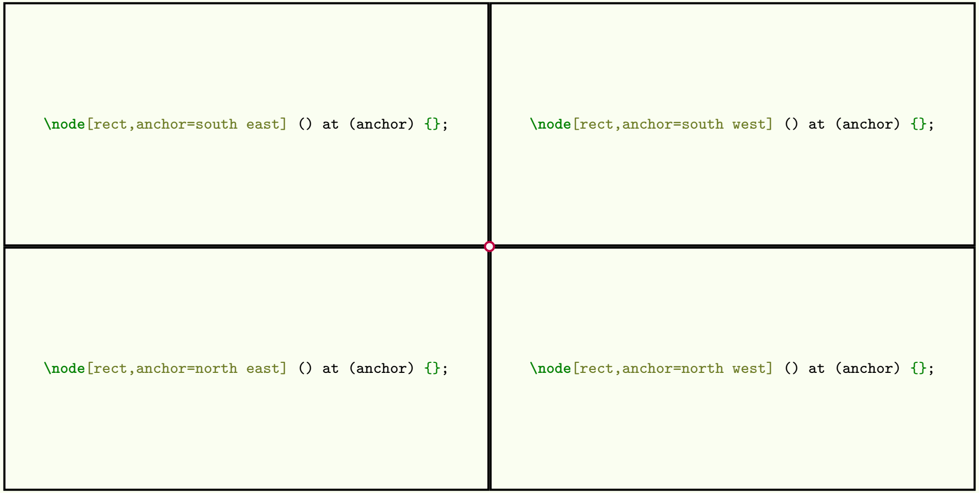




the anchor points (except for `center`) are at the outer edge of the shape

```
\coordinate (anchor) at (current page.center);
```

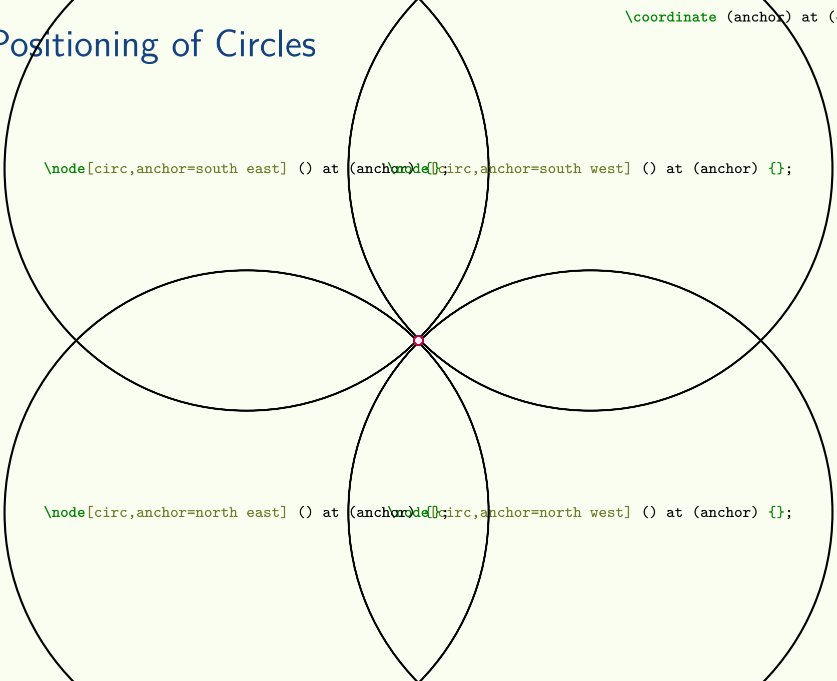
Nodes ▷ Positioning of Rectangles



```
\node[rect,anchor=south east] () at (anchor) {};
\node[rect,anchor=south west] () at (anchor) {};
\node[rect,anchor=north east] () at (anchor) {};
\node[rect,anchor=north west] () at (anchor) {};
```

Nodes ▷ Positioning of Circles

```
\coordinate (anchor) at (current page.center);
```



```
\node[circ,anchor=south east] () at (anchor) {};
\node[circ,anchor=south west] () at (anchor) {};
\node[circ,anchor=north east] () at (anchor) {};
\node[circ,anchor=north west] () at (anchor) {};
```

Nodes ▷ The Page

Within the `tikzpicture` environment one node is defined: “`current page`”.

This node is completely transparent and has the shape and position of the *current page*.

Often, it makes sense to use this node for positioning rather than relying on the current location on the page.

- This makes the positioning absolute (relative to the page) rather than relative (relative to the cursor position).

Coordinates

A coordinate is a convenience construct. It allows you to define a coordinate without a node.

For instance, we can define an offsat center:

```
\coordinate (center) at ([yshift=-12mm] current page.center);
```

And then future positions can be based on this `center` definition instead of something more complex:

```
\node[rectangle,anchor=east] () at ([xshift=-2mm] center) {left};
\node[rectangle,anchor=west] () at ([xshift= 2mm] center) {right};
```

The shift can be in two dimensions:

```
\node[rectangle] () at ([xshift=2mm,yshift=3cm] center) {translation};
```

Paths

A Ti*k*Z path is tightly related to the notion of a path that we know from graph theory.

A Ti*k*Z path is a sequence of edges, when destination of one edge is the same as the source of the following edge.

The individual edges in a path can be drawn according to different rulesets:

- Straight lines.
- 90° bend lines.
- Curves.

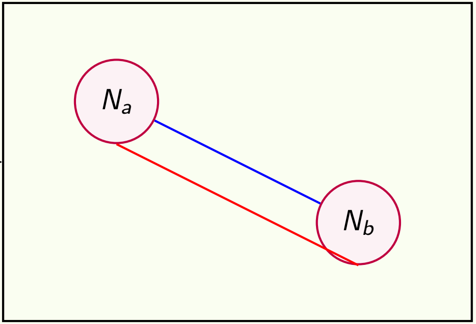
All edges in a path follow the same styling.

Paths ▷ Node vs Points

```
\node[point] (a) at
  ([xshift=-1.6cm,yshift= 8mm]anchor)
  {$N_a$};

\node[point] (b) at
  ([xshift= 1.6cm,yshift=-8mm]anchor)
  {$N_b$};

\draw[edge,draw=blue] (a)
                   -- (b);
\draw[edge,draw=red]  (a.south)
                   -- (b.south);
```

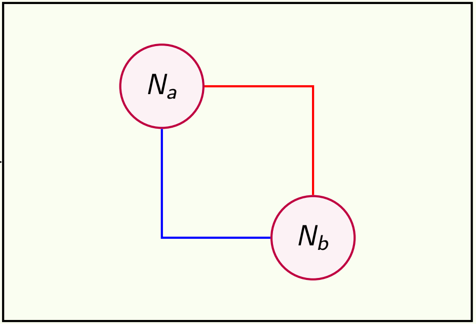


Paths ▷ Angled Edges

```
\node[point] (a)
  at ([xshift=-1cm,yshift= 1cm]anchor)
  {$N_a$};

\node[point] (b)
  at ([xshift= 1cm,yshift=-1cm]anchor)
  {$N_b$};

\draw[edge,draw=red]  (a) -| (b);
\draw[edge,draw=blue] (a) |- (b);
```

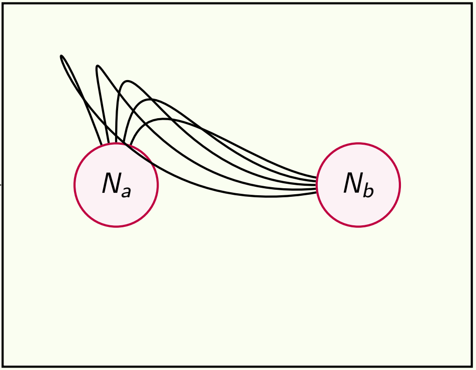


Paths ▷ Curved Edges

```
\newcommand{\aI}[0]{180}
\newcommand{\aO}[0]{90}

\node[point] (a) at ([xshift=-1.6cm]anchor) {$N_a$};
\node[point] (b) at ([xshift= 1.6cm]anchor) {$N_b$};

\draw[edge]
  (a) to[out=\aO+20,in=\aI+10,looseness=3.0] (b);
\draw[edge]
  (a) to[out=\aO+10,in=\aI+ 5,looseness=2.5] (b);
\draw[edge]
  (a) to[out=\aO+ 0,in=\aI+ 0,looseness=2.0] (b);
\draw[edge]
  (a) to[out=\aO-10,in=\aI- 5,looseness=1.5] (b);
\draw[edge]
  (a) to[out=\aO-20,in=\aI-10,looseness=1.0] (b);
```



Styles

Each node and path is drawn according to a sequence of style rules.

Later rules override earlier rules.

We can define shorthands for sequences of rules, and these may refer to other such shorthands.

Format:

```
\tikzstyle{dedge} = [thick,->,>=stealth,draw=black]
```

$$\texttt{\textbackslash tikzstyle\{}\underset{\overset{\uparrow}{\text{name}}}{\boxed{\texttt{dedge}}}\texttt{\}} = \underset{\overset{\uparrow}{\text{Rules}}}{\boxed{\texttt{[thick,->,>=stealth,draw=black]}}}$$

Styles ▷ Select Rules

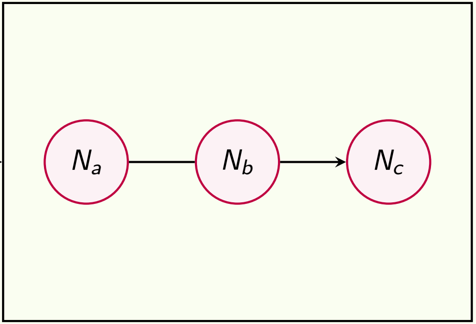
- Name of a previously defined style. Following rules will override.
- Anchor point: `anchor=north west`
- Line thickness: `thin`, `thick`, `very thick`
- Stroke color: `draw=purple`
- Fill color: `fill=purple`
- Text color: `text=purple`
- Text alignment: `align=left`
  (needed for multiple lines)
- Node minimum width: `minimum width=2.4cm`
- Node minimum height: `minimum height=12mm`
- Padding between node text and shape: `inner sep=0pt`

Styles ▷ Use of Derived Styles

```
\tikzstyle{edge} = [thick,draw=black]
\tikzstyle{dedge} = [edge,->,>=stealth]

\node[point] (a)
  at ([xshift=-2cm]anchor) {$N_a$};
\node[point] (b)
  at ([xshift= 0cm]anchor) {$N_b$};
\node[point] (c)
  at ([xshift= 2cm]anchor) {$N_c$};

\draw[edge]  (a)--(b);
\draw[dedge] (b)--(c);
```

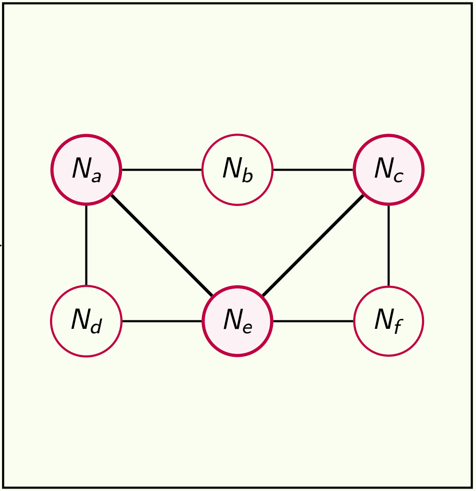


Styles ▷ Use of Multiple Styles

```
\tikzstyle{point}=[
  circle,
  draw=purple,
  thick,
]
\tikzstyle{hl}=[
  fill=purple!5,
  very thick,
]

\node[point,hl] (a)
  at ([xshift=-2cm,yshift= 1cm]anchor) {$N_a$};
\node[point]    (b)
  at ([xshift= 0cm,yshift= 1cm]anchor) {$N_b$};
\node[point,hl] (c)
  at ([xshift= 2cm,yshift= 1cm]anchor) {$N_c$};
\node[point]    (d)
  at ([xshift=-2cm,yshift=-1cm]anchor) {$N_d$};
\node[point,hl] (e)
  at ([xshift= 0cm,yshift=-1cm]anchor) {$N_e$};
\node[point]    (f)
  at ([xshift= 2cm,yshift=-1cm]anchor) {$N_f$};

\draw[edge] (a)--(b)--(c)--(f)--(e)--(d)--(a);
\draw[edge,hl] (a)--(e)--(c);
```



Fancy Stuff

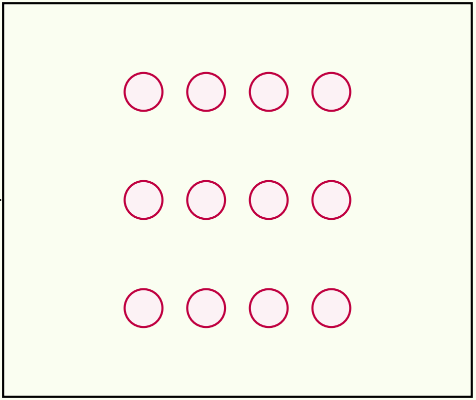
Note that most of the following examples require several Ti*k*Z libraries to be loaded.

The set of libraries I load is ever expanding:

```
\usetikzlibrary[positioning]
\usetikzlibrary[fit]
\usetikzlibrary{patterns}
\usetikzlibrary{patterns.meta}
\usetikzlibrary{shapes.geometric}
\usetikzlibrary{shapes.arrows}
\usetikzlibrary{shapes}
\usetikzlibrary{arrows.meta}
\usetikzlibrary{calc}
\usetikzlibrary{matrix}
\usetikzlibrary{tikzmark}
\usetikzlibrary{datavisualization}
\usetikzlibrary{datavisualization.formats.functions}
```

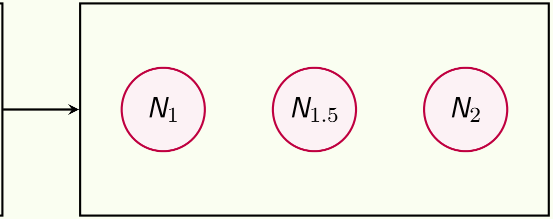
Fancy Stuff ▷ Matrix Alignment

```
\node[matrix,column sep=3mm,row sep=9mm]
  (name) at (anchor) {
    \node[point] (aa) {}; &
    \node[point] (ab) {}; &
    \node[point] (ac) {}; &
    \node[point] (ad) {}; \\
    \node[point] (ba) {}; &
    \node[point] (bb) {}; &
    \node[point] (bc) {}; &
    \node[point] (bd) {}; \\
    \node[point] (ca) {}; &
    \node[point] (cb) {}; &
    \node[point] (cc) {}; &
    \node[point] (cd) {}; \\
};
```



Fancy Stuff ▷ Point Interpolation

```
\node[point] (a)
  at ([xshift=-2cm]anchor) {$N_1$};
\node[point] (b)
  at ([xshift= 2cm]anchor) {$N_2$};
\node[point] (c)
  at ($(a)!0.5!(b)$) {$N_{1.5}$};
```

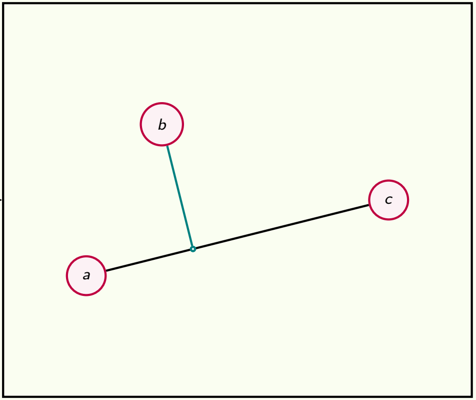


Fancy Stuff ▷ Projection

```
\node[point] (a)
  at ([xshift=-2cm,yshift=-1cm]anchor)
  {$a$};
\node[point] (b)
  at ([xshift=-1cm,yshift= 1cm]anchor)
  {$b$};
\node[point] (c)
  at ([xshift= 2cm]anchor)
  {$c$};
\draw[edge] (a)--(c);

\node[point,hl] (p)
  at ($(a)!(b)!(c)$) {};

\draw[edge,hl] (b)--(p);
```



Part 3:
Automation

Table Heatmaps

A heatmap is a form of visualization whereby the values of the cells of a grid is illustrated by some gradient of colors.

In this section we will look at one application:

We are faced with a choice of data structures. Either we use a list or we use a dictionary (aka map). We will be doing a lot of insertions and want to see how fast each of these operations are, and how well they scale.

Note: In reality this particular problem is likely better solved by looking up a textbook, but it will serve nicely as an example of a type of problem that we might face.

Table Heatmaps ▷ Code

```
#!/usr/bin/env python3

from time import perf_counter

def report(algo, n, time):
  print(",".join([algo, str(n), str(time)]))

def run_list(n):
  l = []
  for i in range(n):
    l.append(i)

def run_dict(n):
  d = {}
  for i in range(n):
    d[i] = [i]

algos = {
  "list append": run_list,
  "dict insert": run_dict,
}

for algo in ["list append", "dict insert"]:
  for i in [4, 8, 12, 16, 20, 24, 28, 32, 36, 40]:
    n = i * 1000 * 1000
    fun = algos[algo]
    t0 = perf_counter()
    fun(n)
    t1 = perf_counter()
    report(algo, n, t1-t0)
```

Table Heatmaps ▷ Benchmarking Results

```
list append,4000000,0.1393402791582048
list append,8000000,0.2268226812593639
list append,12000000,0.34418921219184995
list append,16000000,0.4442757270298898
list append,20000000,0.567413522861898
list append,24000000,0.6672850423492491
list append,28000000,0.8138193367049098
list append,32000000,0.9020816199481487
list append,36000000,1.031110092997551
list append,40000000,1.117086899932474
dict insert,4000000,1.3747214269824326
dict insert,8000000,2.9234286141581833
dict insert,12000000,4.974996192846447
dict insert,16000000,5.923337824176997
dict insert,20000000,7.548715376760811
dict insert,24000000,8.798577990848571
dict insert,28000000,10.580839905887842
dict insert,32000000,13.511960373725742
dict insert,36000000,17.037355119362473
dict insert,40000000,14.978080991189927
```

Table Heatmaps ▷ Processing Code

```python
#!/usr/bin/env python3

import json
from sys import argv

# read input
with open(argv[1]) as fo:
  data = {}
  for line in fo.readlines():
    elems = line.strip().split(",")
    if len(elems)!=3: next

    algo =       elems[0]
    n    =   int(elems[1])
    t    = float(elems[2])

    if not algo in data: data[algo] = {}
    data[algo][n] = t

# find algo, ns, and min and max cell value
algos = []
ns = []
mn = mx = None
for algo in data.keys():
  for n in data[algo]:
    if not algo in algos: algos.append(algo)
    if not n in ns: ns.append(n)

    value = data[algo][n]
    if mn==None or value<mn: mn = value
    if mx==None or value>mx: mx = value

valuemapper = lambda v: (float(v)-mn)/(mx-mn)*100

xcount = len(ns)
ycount = len(algos)

# produce result
yheading = "\\multirow{%i}{*}{\\rotatebox{90}{\\centering
↪  Algorithm}}" % ycount
lines = []
lines.append("\\begin{tabular}{rl%s}" % ("c"*xcount))
lines.append("  & & \\multicolumn{%i}{c}{Number of Operations
↪  / [in 1,000,000]} \\\\" % xcount)
lines.append("  & "+("".join(map(lambda key: " &
↪  "+str(int(key/1000000)), ns)))+" \\\\")
for algo in algos:
  line = "  %s & %s" % (yheading if algo==algos[0] else "",
  ↪  algo)
  for n in sorted(ns):
    value = data[algo][n]
    line += " &\\cellcolor{red!%i!green}%.2f" %
    ↪  (valuemapper(value), value)
  lines.append(line+" \\\\")
lines.append("\\end{tabular}")

# write output
with open(argv[2], "w") as fo:
  fo.writelines("".join(map(lambda line: line+"\n", lines)))
```

Table Heatmaps ▷ Presentation Result

		Number of Operations / [in 1,000,000]									
		4	8	12	16	20	24	28	32	36	40
Algorithm	list append	0.14	0.23	0.34	0.44	0.57	0.67	0.81	0.90	1.03	1.12
	dict insert	1.37	2.92	4.97	5.92	7.55	8.80	10.58	13.51	17.04	14.98

Table Heatmaps ▷ Takeaways

That code can be automatically included in my report by my build system.

Whenever I change `code.py` or `process.py`, the build system will redo the steps necessary to make the resulting PDF reflect that change.

That means:

- The performance numbers will always be true to the latest version of the code (when executed on the build machine).
- The presentation of the data will always be true to the latest version of the processing script.
- Observing these qualities comes at zero cost for the programmer.

Exercise Difficulty

I am writing a book. Each chapter ends with a number of exercises.

You want to score each exercise on two different parameters so that the reader can have a good impression of what they are getting themselves into.

You could hardcode this in every single exercise, but that would be

- hard to manage (as your impression of the true exercise difficulty changes over time), and
- hard to keep consistent.

Exercise Difficulty ▷ Solution

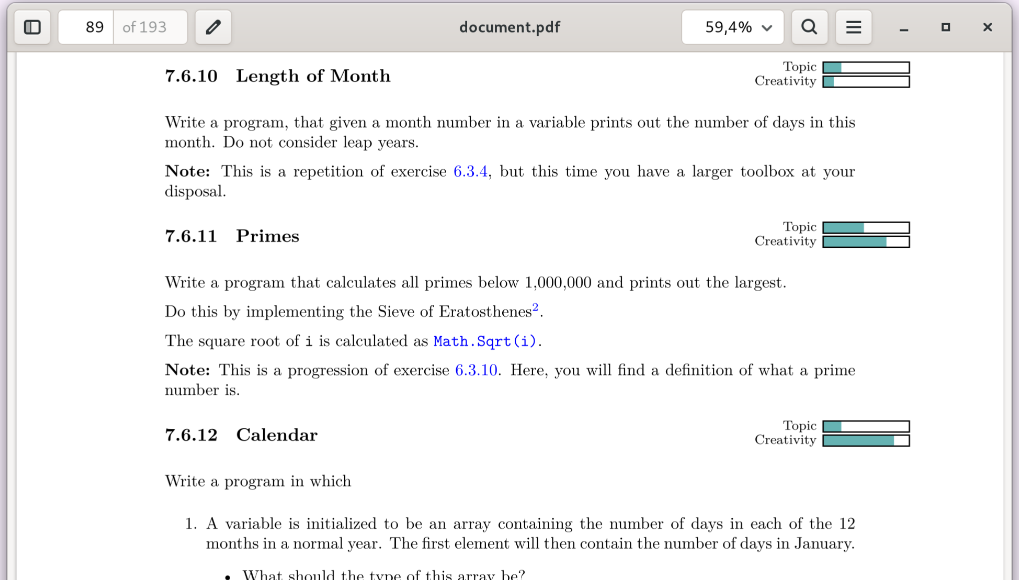
First, introduce directory structure:

- Chapter name at the top level.
- Some sort of machine readable index file in these directories (e.g., YAML).
- Exercise name at the next level.
- At the last level we have a `question.tex`, a `answer.tex` and a `meta.yaml` (with the difficulties).

Next, we write a script that processes that part of the filesystem and produces an exercise file per chapter. Those exercise files are then included in the main document.

Now that we know Ti*k*Z, and we have everything available to our script, we might as well make it pretty ...

Exercise Difficulty ▷ Result



89 of 193

document.pdf

59,4%

7.6.10 Length of Month

Write a program, that given a month number in a variable prints out the number of days in this month. Do not consider leap years.

Note: This is a repetition of exercise 6.3.4, but this time you have a larger toolbox at your disposal.

7.6.11 Primes

Write a program that calculates all primes below 1,000,000 and prints out the largest.

Do this by implementing the Sieve of Eratosthenes[2].

The square root of `i` is calculated as `Math.Sqrt(i)`.

Note: This is a progression of exercise 6.3.10. Here, you will find a definition of what a prime number is.

7.6.12 Calendar

Write a program in which

1. A variable is initialized to be an array containing the number of days in each of the 12 months in a normal year. The first element will then contain the number of days in January.
   - What should the type of this array be?

Nametag Generation

You got a spreadsheet with 200+ registrations for a conference, and you need to produce nametags for everyone.

Participants have registered for a subset of the planned events which are spread out throughout the course of a (work)week.

The information that needs to find their way to each individual nametag is spread out across 60+ columns in a spreadsheet, and the mapping is far from clean.

A reference to the full conference program should be placed on each nametag.

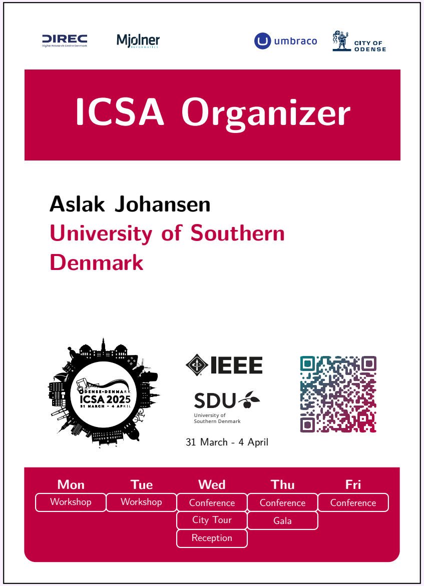
You have logos for the conference and sponsors.

Nametag Generation ▷ Solution

Write a script that loads the spreadsheet, crunches the data and writes a `.tex` file that makes use of a set of definitions from another `.tex` file.

Source: `https://github.com/aslakjohansen/icsa25-nametag`

Nametag Generation ▷ Result



DIREC Mjolner umbraco CITY OF ODENSE

ICSA Organizer

Aslak Johansen
University of Southern Denmark

ICSA 2025 ODENSE·DENMARK 31 MARCH - 4 APRIL

IEEE

SDU University of Southern Denmark

31 March - 4 April

Mon	Tue	Wed	Thu	Fri
Workshop	Workshop	Conference	Conference	Conference
		City Tour	Gala	
		Reception		

Parameterized Documents

You need to produce a large number (n) of documents that has a lot in common.

They each have a different title.

You already have one file with document-specific contents. So, n files here.

Some content is the same. So, 1 file here.

The *naive* solution is to then have n more files, essentially doing:

1. Set title.
2. Include common file.
3. Include specific file.

This, however, can be done using 1 file and a build system.

Parameterized Documents ▷ Solution

The LaTeX engine command is usually given a filename as parameter:

```
lualatex -shell-escape intro_presentation.tex
```

It is the contents of this file that it tries to produce a PDF file from.

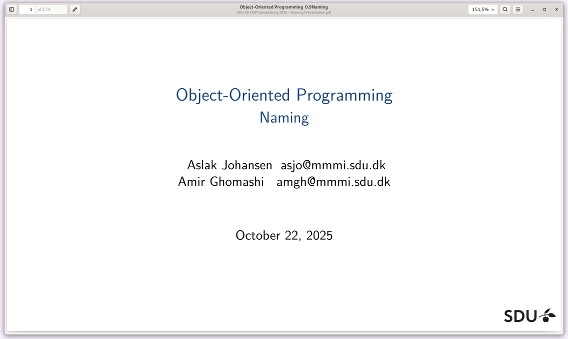
There is, however, another way to provide it with LaTeX code:

```
lualatex -shell-escape "\newcommand\odenseonly[1]{\#1}
↪  \newcommand\sonderborgonly[1]{} \input{intro_presentation.tex}"
```

So, what happens here?

- Two macros a produced: One that drops its parameter and one that retains it.
- Then the rest of the code is loaded. This code relies on the two macros.
- In another command the functionality of the two macros is flipped.

Parameterized Documents ▷ Result



1 of 174

Object-Oriented Programming 0.9Naming

SDU SE OOP Sønderborg 2025 - Naming Presentation.pdf

151,5%

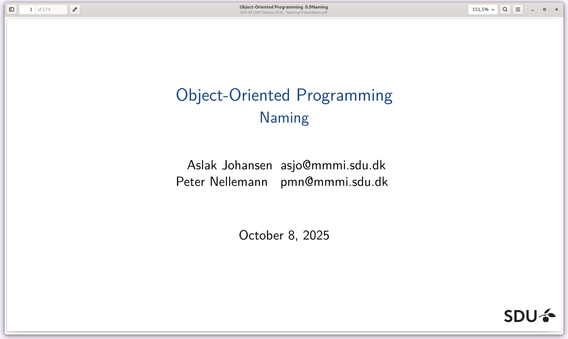
Object-Oriented Programming

Naming

Aslak Johansen asjo@mmmi.sdu.dk
Amir Ghomashi amgh@mmmi.sdu.dk

October 22, 2025

SDU



1 of 174

Object-Oriented Programming 0.9Naming

SDU SE OOP Odense 2025 - Naming Presentation.pdf

151,5%

Object-Oriented Programming

Naming

Aslak Johansen asjo@mmmi.sdu.dk
Peter Nellemann pmn@mmmi.sdu.dk

October 8, 2025

SDU

Graph Layout

You have to include a visual representation of a graph.

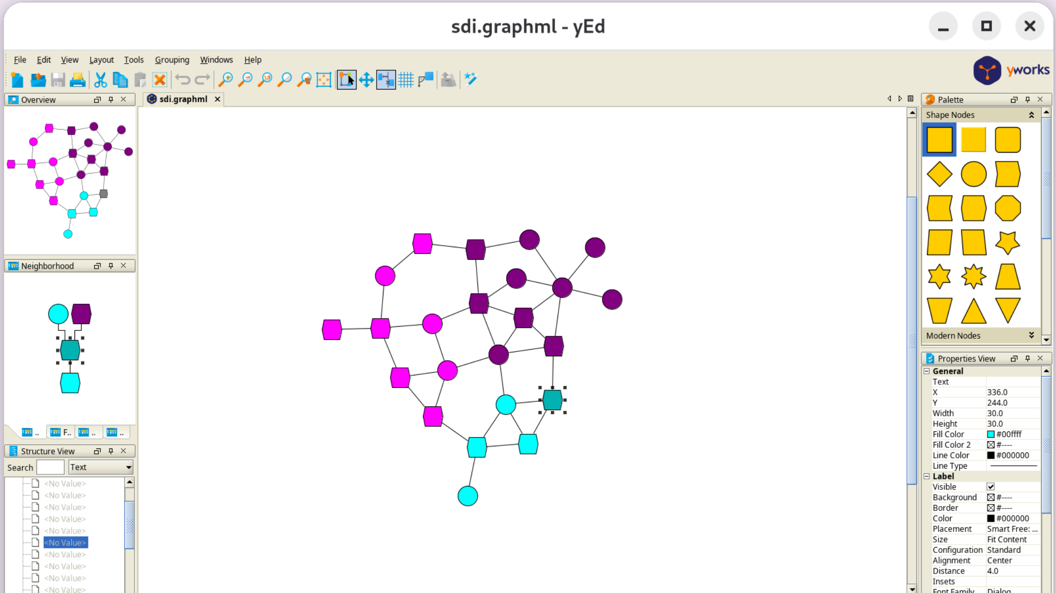
It has a number of different properties that can be mapped to color, shape ...

It has a significant number of nodes and connections. This makes the job of layouting the graph non-trivial.

Manual layouting is likely to have horribly looking results.

The **yEd** program is quite good at layouting and can work with XML files.

Graph Layout ▷ yEd



General	
Text	
X	336.0
Y	244.0
Width	30.0
Height	30.0
Fill Color	🟦 #00ffff
Fill Color 2	☒ #----
Line Color	■ #000000
Line Type	
Label	
Visible	✓
Background	☒ #----
Border	☒ #----
Color	■ #000000
Placement	Smart Free: ...
Size	Fit Content
Configuration	Standard
Alignment	Center
Distance	4.0
Insets	
Font Family	Dialog

Graph Layout ▷ Nodes in GraphML

```
<node id="n3">
  <data key="d5"/>
  <data key="d6">
    <y:ShapeNode>
      <y:Geometry height="30.0" width="30.0" x="104.0" y="210.0"/>
      <y:Fill color="#FF00FF" transparent="false"/>
      <y:BorderStyle color="#000000" raised="false" type="line" width="1.0"/>
      <y:NodeLabel alignment="center" autoSizePolicy="content" fontFamily="Dialog"
          fontSize="12" fontStyle="plain" hasBackgroundColor="false" hasLineColor="false"
          hasText="false" height="4.0" horizontalTextPosition="center" iconTextGap="4"
          modelName="custom" textColor="#000000" verticalTextPosition="bottom" visible="true"
          width="4.0" x="13.0" y="13.0">
        <y:LabelModel>
          <y:SmartNodeLabelModel distance="4.0"/>
        </y:LabelModel>
        <y:ModelParameter>
          <y:SmartNodeLabelModelParameter labelRatioX="0.0" labelRatioY="0.0" nodeRatioX="0.0"
              nodeRatioY="0.0" offsetX="0.0" offsetY="0.0" upX="0.0" upY="-1.0"/>
        </y:ModelParameter>
      </y:NodeLabel>
      <y:Shape type="hexagon"/>
    </y:ShapeNode>
  </data>
</node>
```

Graph Layout ▷ Edges in GraphML

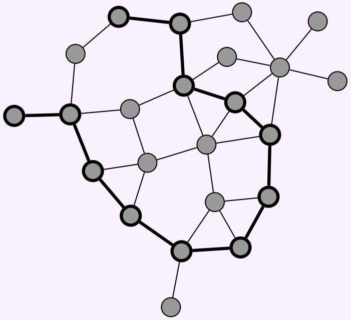
```
<edge id="e2" source="n2" target="n3">
  <data key="d9"/>
  <data key="d10">
    <y:PolyLineEdge>
      <y:Path sx="0.0" sy="0.0" tx="0.0" ty="0.0"/>
      <y:LineStyle color="#000000" type="line" width="1.0"/>
      <y:Arrows source="none" target="none"/>
      <y:BendStyle smoothed="false"/>
    </y:PolyLineEdge>
  </data>
</edge>
```

Graph Layout ▷ Processing Code

1. Define regular expressions for matching relevant lines and extracting needed values.
2. Parse the GraphML file using the regular expressions.
   Note: This is obviously <u>so</u> the wrong approach ... but it works.
3. Find min and max positions on both the x and y axes, and use these to define scaling functions.
4. Make a string containing Ti*k*Z code:
   4.1 Add definitions of styles.
   4.2 Add code for each node.
      - Apply scalers from step 3.
      - Reuse id's from GraphML file.

   4.3 Add code for each edge.
5. Save that string to a file.

Graph Layout ▷ Result



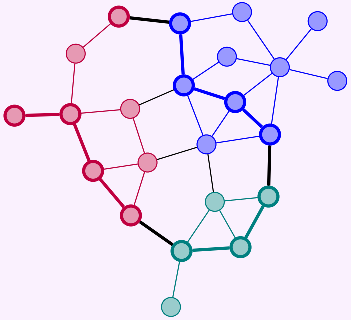


Figure Narratives

You have to make a presentation where:

- There are complex graphical elements.
- Throughout the presentation some of these elements have to appear and/or disappear.
- Throughout the presentation some of these elements have to change appearance (e.g., change color or thickness).

You want to use a GUI to produce the base figure, and then automatically derive the needed variants.

Figure Narratives ▷ Solution Pipeline

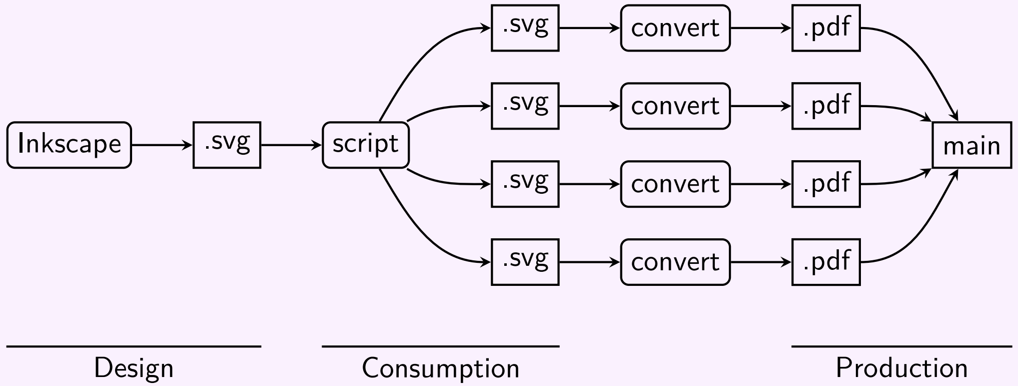
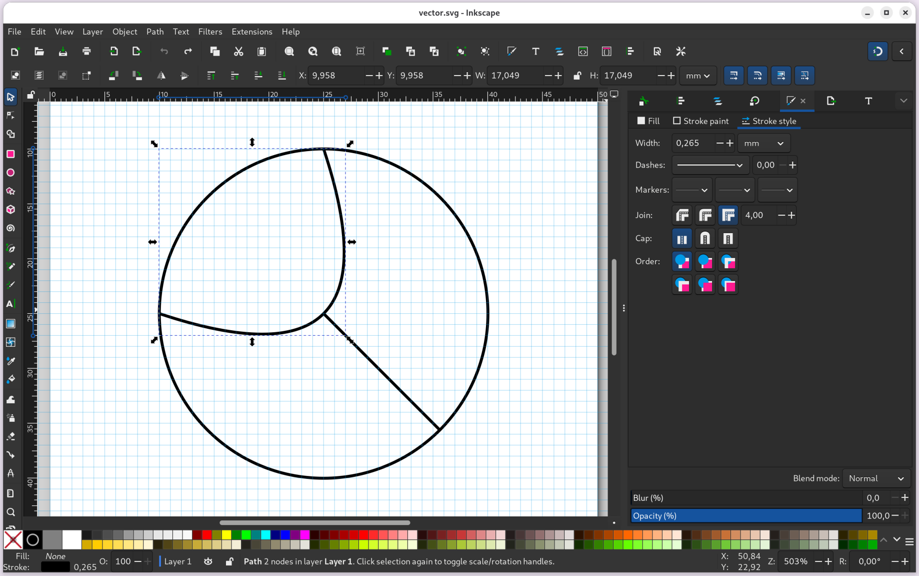


Figure Narratives ▷ Drawing with Inkscape



X: 9,958 Y: 9,958 W: 17,049 H: 17,049 mm

Width: 0,265 mm

Dashes: 0,00

Markers:

Join: 4,00

Cap:

Order:

Blend mode: Normal

Blur (%) 0,0

Opacity (%) 100,0

Fill: None

Stroke: 0,265

O: 100

Layer 1

Path 2 nodes in layer **Layer 1**. Click selection again to toggle scale/rotation handles.

X: 50,84 Y: 22,92 Z: 503% R: 0,00°

Figure Narratives ▷ Base File

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!-- Created with Inkscape (http://www.inkscape.org/) -->

<svg
   width="50mm"
   height="50mm"
   viewBox="0 0 50 50"
   id="SVGRoot"
   version="1.1"
   inkscape:version="1.4 (e7c3feb100, 2024-10-09)"
   sodipodi:docname="vector.svg"
   xmlns:inkscape="http://www.inkscape.org/namespaces/inkscape"
   xmlns:sodipodi="http://sodipodi.sourceforge.net/DTD/sodipodi-0.dtd"
   xmlns="http://www.w3.org/2000/svg"
   xmlns:svg="http://www.w3.org/2000/svg">
  <sodipodi:namedview
     inkscape:document-units="mm"
     inkscape:zoom="5.0329577"
     inkscape:cx="94.278559"
     inkscape:cy="107.09409"
     id="namedview1"
     pagecolor="#ffffff"
     bordercolor="#000000"
     borderopacity="0.25"
     inkscape:showpageshadow="2"
     inkscape:pageopacity="0.0"
     inkscape:pagecheckerboard="0"
     inkscape:deskcolor="#d1d1d1"
     inkscape:window-width="1920"
     inkscape:window-height="1131"
     inkscape:window-x="0"
     inkscape:window-y="32"
     inkscape:window-maximized="1"
     inkscape:current-layer="layer1"
     showgrid="true">
    <inkscape:grid
       id="grid1"
       units="mm"
       originx="0"
       originy="0"
       spacingx="0.1"
       spacingy="0.1"
       empcolor="#0099e5"
       empopacity="0.30196078"
       color="#0099e5"
       opacity="0.14901961"
       empspacing="10"
       enabled="true"
       visible="true" />
  </sodipodi:namedview>
  <defs
     id="defs1" />
  <g
     inkscape:label="Layer 1"
     inkscape:groupmode="layer"
     id="layer1">
    <ellipse
       style="fill:none;stroke:#000000;stroke-width:0.264583"
       id="path3"
       cx="24.999998"
       cy="25"
       rx="14.999999"
       ry="15" />
    <path
       style="fill:none;stroke:#000000;stroke-width:0.264583"
       d="m 10,25 c 15,5 20,0 15,-15"
       id="path4"
       sodipodi:nodetypes="cc" />
    <path
       style="fill:#ffffff;stroke:#000000;stroke-width:0.264583"
       d="M 25,25 35.600001,35.600001"
       id="path5"
       sodipodi:nodetypes="cc" />
  </g>
</svg>
```

Figure Narratives ▷ Processing Code

```python
#!/usr/bin/env python3

from svgnarrative import Model

#############################################################
################################################# definitions

i = 0
m = Model('figs/vector.svg')

circle = "path3"
curve  = "path4"
line   = "path5"
group  = "layer1"

#############################################################
##################################################### helpers

def store ():
  global i
  filename_svg = 'figs/vector%d.svg' % i
  m.store(filename_svg)
  i += 1

#############################################################
######################################################## main
```

```python
# clean start
m.hide(circle)
m.hide(curve)
m.hide(line)
store()

# reveal circle
m.show(circle)
store()

# reveal curve
m.show(curve)
store()

# reveal line
m.show(line)
store()

# hide everything through outer group (elements still visible)
m.hide(group)
store()

# add some color
m.stroke(circle, "#ff8800")
m.show(group)
store()
```

Figure Narratives ▷ Conversion to PDF and Inclusion

This process relies on the following macro definition:

```
\newcommand{\includeSVG}[1]{\includegraphics[scale=1.0]{./figs/#1.pdf}}
```

In the document text, I can then insert the figure like this:

```
\includeSVG{vector5}
```

And then have a script crawl through the file inclusion tree of the root LaTeX document to find all references to this macro, and automatically convert each of them from SVG to PDF.

Figure Narratives ▷ Result

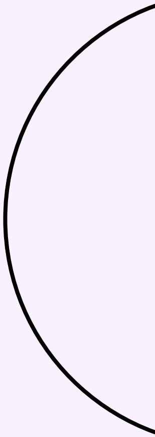


Figure Narratives ▷ Tooling

This is something that I have had a need for on some number of occasions.

Because of that, I wrote the `svgnarrative` python package.

It is available on PIP:

https://pypi.org/project/svgnarrative/

And on GitHub:

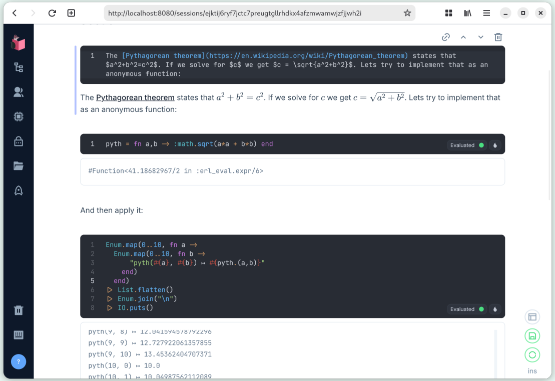
https://github.com/aslakjohansen/svg-narrative/

Part 4:
Next Steps

KaTeX

KaTeX allows LaTeX to be used on the web, and sometimes through *markdown*.

More info: https://katex.org



http://localhost:8080/sessions/ejktij6ryf7jctc7preugtgllrhdkx4afzmwamwjzfjjwh2i

```
The [Pythagorean theorem](https://en.wikipedia.org/wiki/Pythagorean_theorem) states that
$a^2+b^2=c^2$. If we solve for $c$ we get $c = \sqrt{a^2+b^2}$. Lets try to implement that as an
anonymous function:
```

The Pythagorean theorem states that $a^2 + b^2 = c^2$. If we solve for c we get $c = \sqrt{a^2 + b^2}$. Lets try to implement that as an anonymous function:

```
pyth = fn a,b → :math.sqrt(a*a + b*b) end
```

```
#Function<41.18682967/2 in :erl_eval.expr/6>
```

And then apply it:

```
Enum.map(0..10, fn a →
  Enum.map(0..10, fn b →
      "pyth(#{a}, #{b}) ↦ #{pyth.(a,b)}"
    end)
  end)
▷ List.flatten()
▷ Enum.join("\n")
▷ IO.puts()
```

```
pyth(9, 8) ↦ 12.041594578792296
pyth(9, 9) ↦ 12.727922061357855
pyth(9, 10) ↦ 13.45362404707371
pyth(10, 0) ↦ 10.0
pyth(10, 1) ↦ 10.04987562112089
```

Animation Engines

3Blue1Brown:

- Homepage: `https://www.3blue1brown.com`
- Youtube: `https://www.youtube.com/@3blue1brown`

Versions:

- Manim:
  - Github: `https://github.com/3b1b/manim`
- Manim CE:
  - Github: `https://github.com/ManimCommunity/manim`

Next Level

Proper TeX programming: State, loops, state ...

Constructive methodology for writing LaTeX and dealing with errors.

Defining custom Ti*k*Z node types, edge types and anchor points.

Writing style.

Part 5: Exercises

Exercises

1. ***Color Cube:*** Use Ti*k*Z to draw an RGB color cube that is centered on the page. It should have nodes for red, green, blue, black, cyan, magenta, yellow and white, and the appropriate edges.
2. ***First Macro:*** Lets imagine that you are working on a project where you deal with objects that have a color property. For some reason it is important for you, in a report, to be able to quickly state whether one object is more or less red, green and blue than another. Write a number of macros, that make it easy to write something like this:

$$O_1 \overset{\color{red}\bullet}{=} O_2 \qquad\qquad O_1 \overset{\color{green}\bullet}{<} O_2 \qquad\qquad O_1 \overset{\color{blue}\bullet}{\geq} O_2$$

3. ***Automated Test Reporting*** Imagine that you have a test suite (e.g., unit tests) and you have to report the outcome as part of the documentation of your work. Pick any test suite that you have lying around. Implement a pipeline that, as part of the build process of a LaTeX document, makes sure to run the tests, parse the results and produce a table with the results that is included in the document.