The Ti*k*Z-Extensions Package

Manual for version 0.1

Qrrbrbirlbel

August 14, 2022

Contents

Part I
Introduction

1 Usage

This package is called `tikz-ext`, however, one does *not* load it via `\usepackage{tikz-ext}`. Instead, this package consists of multiple PGF and Ti*k*Z libraries which are loaded by either `\usepgflibrary` or `\usetikzlibrary`.

2 Why do we need it?

Since I have been answering questions on TeX.sx I've noticed that some questions come up again and again, every time with a slightly different approach on how to solve them.

I don't like reinventing the wheel which is why I've gathered the code of my answers in this package.

And, yes, I am using them myself, too.

3 Should these libraries be part of Ti*k*Z?

I guess.

Part II
Ti*k*Z Libraries

4 Extending the Path Timers

Ti*k*Z Library `paths.timer`

```
\usetikzlibrary{paths.timer} % LaTeX and plain TeX
\usetikzlibrary[paths.timer] % ConTeXt
```

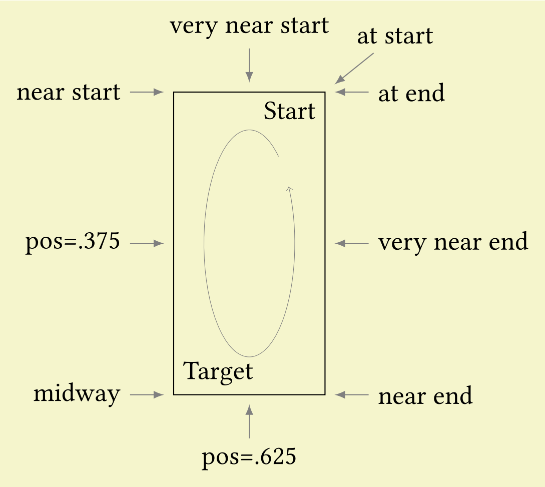
This library adds timers to the path specifications `rectangle`, `parabola`, `sin` and `cos`.

In Ti*k*Z, the path specification `rectangle`, `parabola`, `sin` and `cos` do not provide their own timer, i. e. a node placing algorithm that is dependent on the actual path. For `rectangle` the timer of the straight line between the rectangle's corners is used, for the other paths, nodes, coordinates, pics, etc. are placed on the last coordinate.

This library allows this.

4.1 Rectangle

For the `rectangle` path operator, the timer starts with `pos = 0` (= `at start`) from the starting coordinate in a counter-clockwise direction along the rectangle. The corners will be at positions 0.0, 0.25, 0.5, 0.75 and 1.0.

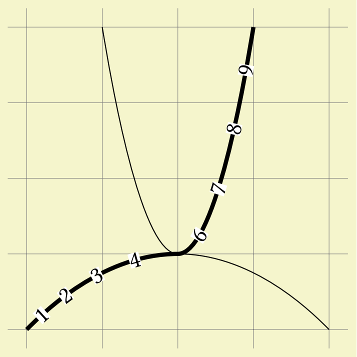


```
\usetikzlibrary {paths.timer}
\begin{tikzpicture}[scale=2, every pin edge/.style={latex-, gray}]
\coordinate [label=above right:Target] (A) at (0,0);
\coordinate [label=below left:Start]   (B) at (1,2);
\draw[->, help lines] ([shift=(50:.3 and .75)] .5,1)
  arc[start angle=50, delta angle=340, x radius=.3, y radius=.75];
\draw (B) rectangle (A)
  foreach \pos/\ang in {at start/60, very near start/90, near start/180, pos=.375/180,
                        midway/180, pos=.625/270, near end/0, very near end/0, at end/0}{
    node[pin=\ang:\pos, style/.expanded=\pos]{}};
\end{tikzpicture}
```

4.2 Parabola

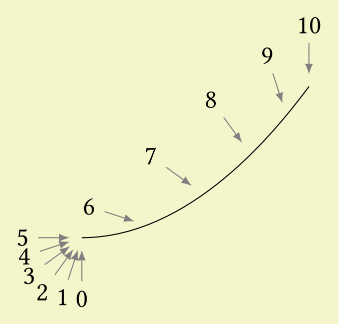
For the `parabola` path operator the timer is similar to the `.. controls ..` operator.

The position 0.5 will lie at the `bend`.



```
\usetikzlibrary {paths.timer}
\begin{tikzpicture}
\draw[help lines]  (-2.25, -1.25) grid (2.25, 3.25);
\draw              ( 2,-1) parabola bend (0,0) (-1,3);
\draw[ultra thick] (-2,-1) parabola bend (0,0) ( 1,3)
  foreach \pos in {1,...,4,6,7,...,9}{
    node[
      pos=.\pos, sloped, fill=white, font=\small, inner sep=+0pt
    ] {\pos}
  };
\end{tikzpicture}
```

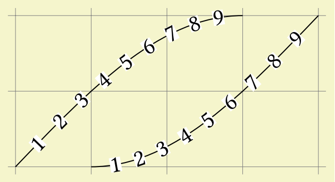
If no `bend` is specified half the positions will collapse into one end of the curve.



```
\usetikzlibrary {paths.timer}
\begin{tikzpicture}[every pin edge/.style={latex-, shorten <=1pt, gray}]
\draw (-2,-2) parabola (1,0)
  foreach \pos in {0, 1, ..., 10} {
    node [pos=\pos/10, pin={[anchor=-18*\pos+90]-18*\pos+270:\pos}]{}
  };
\end{tikzpicture}
```

4.3 Sine/Cosine

The `sin` and `cos` path operators also allow placing of nodes along their paths.



```
\usetikzlibrary {paths.timer}
\begin{tikzpicture}[mark nodes on line/.style={insert path={
  foreach \pos in {1, ..., 9} {node[
    sloped, fill=white, font=\small, inner sep=+0pt, pos=\pos/10] {\pos}}}}]
\draw[help lines] (-2.1,-2.1) grid (2.1,0.1);
\draw             (-2,-2) sin (1,0) [mark nodes on line];
\draw[shift=(0:1)](-2,-2) cos (1,0) [mark nodes on line];
\end{tikzpicture}
```

5 Mirror, Mirror on the Wall

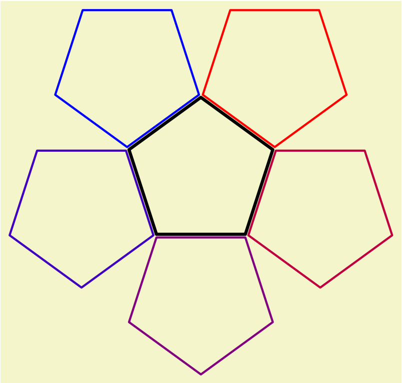
Ti*k*Z Library `transformations.mirror`

```
\usetikzlibrary{transformations.mirror} % LaTeX and plain TeX
\usetikzlibrary[transformations.mirror] % ConTeXt
```

This library adds more transformations to Ti*k*Z.

As explained in section 7, they are two approaches to setting a mirror transformation. As with the commands in PGF, we'll be using lowercase `m` for the "Spiegelungsmatrix" and uppercase `M` for the built-in approach.

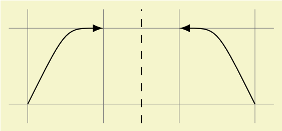
5.1 Using the "Spiegelungsmatrix"



```
\usetikzlibrary {shapes.geometric,transformations.mirror}
\begin{tikzpicture}[line join=round, thick, reg poly/.style={
  shape=regular polygon, regular polygon sides={#1}}]
\node[reg poly=5, minimum size=+2cm, draw, very thick] (a) {};
\foreach \i[evaluate={\col=(\i-1)/.04}] in {1,...,5}
  \node [mirror=(a.corner \i)--(a.side \i), transform shape,
         reg poly=5, minimum size=+2cm, draw=red!\col!blue] {};
\end{tikzpicture}
```

`/tikz/xmirror=`⟨*value or coordinate*⟩ (no default)

Sets up a transformation that mirrors along a horizontal line that goes through point (⟨*value*⟩, 0) or ⟨*coordinate*⟩.



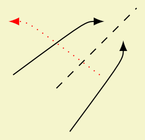
```
\usetikzlibrary {transformations.mirror}
\begin{tikzpicture}
\draw[help lines] (-0.25, -.25) grid (3.25, 1.25);
\draw[-latex] (0,0) .. controls (.5,1) .. (1,1);

\draw[dashed] (1.5, -.25) coordinate (m) -- (1.5, 1.25);
\draw[xmirror=(m),-latex] (0,0) .. controls (.5,1) .. (1,1);
\end{tikzpicture}
```

/tikz/ymirror=⟨*value or coordinate*⟩ (no default)

Sets up a transformation that mirrors along a vertical line that goes through point (0, ⟨*value*⟩) or ⟨*coordinate*⟩.

/tikz/mirror x=⟨*coordinate*⟩ (no default)

Similar to /tikz/xmirror, this however uses the xyz coordinate system instead of the canvas system.



```
\usetikzlibrary {transformations.mirror}
\begin{tikzpicture}[x=.5cm, y=(45:1cm)]

\draw[-latex] (0,0) .. controls (.5,1) .. (1,1);

\draw[dashed] (1.5, -.25) coordinate (m) -- (1.5, 1.25);

\draw[ xmirror=(m), -latex, red, dotted] (0,0) .. controls (.5,1) .. (1,1);
\draw[mirror x=(m), -latex]              (0,0) .. controls (.5,1) .. (1,1);
\end{tikzpicture}
```

/tikz/mirror y=⟨*coordinate*⟩ (no default)

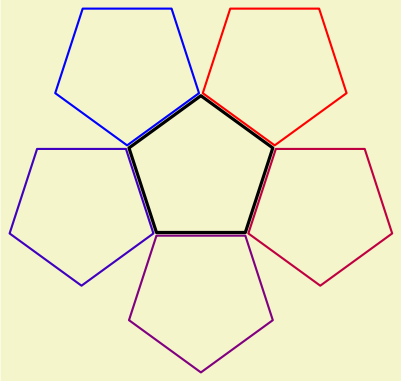
Similar to /tikz/ymirror, this however uses the xyz coordinate system instead of the canvas system.

/tikz/mirror=⟨*point A*⟩--⟨*point B*⟩ (no default)

Sets up a transformation that mirrors along a line that goes through ⟨*point A*⟩ and ⟨*point B*⟩.

When only ⟨*point A*⟩ is given that line goes through ⟨*point A*⟩ and the origin.

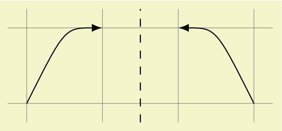
5.2 Using built-in transformations



```
\usetikzlibrary {shapes.geometric,transformations.mirror}
\begin{tikzpicture}[line join=round, thick, reg poly/.style={
  shape=regular polygon, regular polygon sides={#1}}]
\node[reg poly=5, minimum size=+2cm, draw, very thick] (a) {};
\foreach \i[evaluate={\col=(\i-1)/.04}] in {1,...,5}
  \node [Mirror=(a.corner \i)--(a.side \i), transform shape,
         reg poly=5, minimum size=+2cm, draw=red!\col!blue] {};
\end{tikzpicture}
```

`/tikz/xMirror=`⟨*value or coordinate*⟩ (no default)

Sets up a transformation that mirrors along a horizontal line that goes through point (⟨*value*⟩, 0) or ⟨*coordinate*⟩.



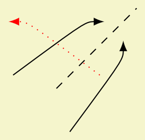
```
\usetikzlibrary {transformations.mirror}
\begin{tikzpicture}
\draw[help lines] (-0.25, -.25) grid (3.25, 1.25);
\draw[-latex] (0,0) .. controls (.5,1) .. (1,1);

\draw[dashed] (1.5, -.25) coordinate (m) -- (1.5, 1.25);
\draw[xMirror=(m),-latex] (0,0) .. controls (.5,1) .. (1,1);
\end{tikzpicture}
```

`/tikz/yMirror=`⟨*value or coordinate*⟩ (no default)

Sets up a transformation that mirrors along a vertical line that goes through point (0, ⟨*value*⟩) or ⟨*coordinate*⟩.

`/tikz/Mirror x=`⟨*coordinate*⟩ (no default)

Similar to `/tikz/xMirror`, this however uses the `xyz` coordinate system instead of the `canvas` system.



```
\usetikzlibrary {transformations.mirror}
\begin{tikzpicture}[x=.5cm, y=(45:1cm)]

\draw[-latex] (0,0) .. controls (.5,1) .. (1,1);

\draw[dashed] (1.5, -.25) coordinate (m) -- (1.5, 1.25);

\draw[ xMirror=(m), -latex, red, dotted] (0,0) .. controls (.5,1) .. (1,1);
\draw[Mirror x=(m), -latex]              (0,0) .. controls (.5,1) .. (1,1);
\end{tikzpicture}
```

`/tikz/Mirror y=`⟨*coordinate*⟩ (no default)

Similar to `/tikz/yMirror`, this however uses the `xyz` coordinate system instead of the `canvas` system.

`/tikz/Mirror=`⟨*point A*⟩`--`⟨*point B*⟩ (no default)

Sets up a transformation that mirrors along a line that goes through ⟨*point A*⟩ and ⟨*point B*⟩.

When only ⟨*point A*⟩ is given that line goes through ⟨*point A*⟩ and the origin.

6 Using Images as a Pattern

Ti*k*Z Library `patterns.images`

```
\usetikzlibrary{patterns.images} % LaTeX and plain TeX
\usetikzlibrary[patterns.images] % ConTeXt
```

This library allows to use an image to be used as a repeating pattern for a path.

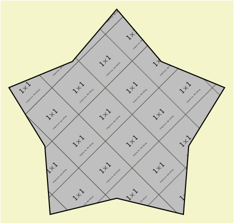
With this library arbitrary images (or indeed PDF documents) can be used as a repeating pattern for the background of a path. This is a two-step process:

1. Declaring an image as an “image-pattern”.

2. Using the “image-pattern”.

`\pgfsetupimageaspattern`[⟨*options*⟩]{⟨*name*⟩}{⟨*image*⟩}

`/tikz/image as pattern=`⟨*options*⟩ (default {})



```
\usetikzlibrary {patterns.images}
\pgfsetupimageaspattern[width=.5cm]{grid}{example-image-1x1}
\tikz \node[star, minimum size=3cm, draw,
  image as pattern={name=grid,options={left, bottom, y=-.5cm, rotate=45}}] {};
```

`/tikz/image as pattern/name=`⟨*name*⟩ (no default)

Specifies the name of the “image-pattern” to be used.

`/tikz/image as pattern/option` (style, no value)

Options that’s be used by the internal `\pgftext`, only keys from `/pgf/text` should be used.

`/tikz/image as pattern/options=`⟨*style*⟩ (style, no default)

Appends style `/tikz/image as pattern/option`.

Part III

PGF Libraries

7 Transformations: Mirroring

Ti*k*Z Library `transformations.mirror`

```
\usepgflibrary{transformations.mirror}  % LaTeX and plain TeX and pure pgf
\usepgflibrary[transformations.mirror]  % ConTeXt and pure pgf
\usetikzlibrary{transformations.mirror} % LaTeX and plain TeX when using TikZ
\usetikzlibrary[transformations.mirror] % ConTeXt when using TikZ
```

This library adds mirror transformations to PGF.

Two approaches to mirror transformation exist:

1. Using the “Spiegelmatrix“ (see section 7.1).

   This depends on `\pgfpointnormalised` which involves the sine and the cosine functions of PGFmath.

2. Using built-in transformations (see section 7.2).

   This depends on `\pgfmathanglebetween` which involves the arctangent (`atan2`) function of PGFmath.

Which one is better? I don’t know. Choose one you’re comfortable with.

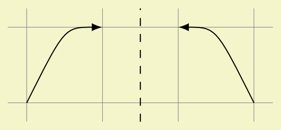
7.1 Using the “Spiegelungsmatrix”

The following commands use the “Spiegelungsmatrix” that sets the transformation matrix following

$$A = \frac{1}{\|\vec{l}\|^2} \begin{bmatrix} l_x^2 - l_y^2 & 2l_xl_y \\ 2l_xl_y & l_y^2 - l_x^2 \end{bmatrix}.$$

`\pgftransformxmirror{`⟨*value*⟩`}`

Sets up a transformation that mirrors along a vertical line that goes through point (⟨*value*⟩, 0).



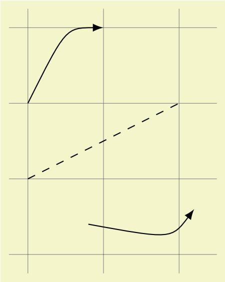
```
\usepgflibrary {transformations.mirror}
\begin{tikzpicture}
\draw[help lines] (-0.25, -.25) grid (3.25, 1.25);
\draw[-latex] (0,0) .. controls (.5,1) .. (1,1);

\draw[dashed] (1.5, -.25) -- (1.5, 1.25);
\pgftransformxmirror{1.5}

\draw[-latex] (0,0) .. controls (.5,1) .. (1,1);
\end{tikzpicture}
```

\pgftransformymirror{⟨*value*⟩}

Sets up a transformation that mirrors along a horizontal line that goes through point $(0, \langle \textit{value} \rangle)$.

\pgftransformmirror{⟨*point A*⟩}{⟨*point B*⟩}

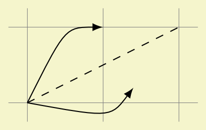
Sets up a transformation that mirrors along the line that goes through ⟨*point A*⟩ and ⟨*point B*⟩.



```
\usepgflibrary {transformations.mirror}
\begin{tikzpicture}
\draw[help lines] (-.25, -2.25) grid (2.5, 1.25);
\draw[-latex] (0,0) .. controls (.5,1) .. (1,1);

\draw[dashed] (0, -1) -- (2, 0);
\pgftransformmirror{\pgfpointxy{0}{-1}}{\pgfpointxy{2}{0}}

\draw[-latex] (0,0) .. controls (.5,1) .. (1,1);
\end{tikzpicture}
```

\pgfqtransformmirror{⟨*point A*⟩}

Sets up a transformation that mirrors along the line that goes through the origin and ⟨*point A*⟩.



```
\usepgflibrary {transformations.mirror}
\begin{tikzpicture}
\draw[help lines] (-.25, -.25) grid (2.25, 1.25);
\draw[-latex] (0,0) .. controls (.5,1) .. (1,1);

\draw[dashed] (0, 0) -- (2, 1);
\pgfqtransformmirror{\pgfpointxy{2}{1}}

\draw[-latex] (0,0) .. controls (.5,1) .. (1,1);
\end{tikzpicture}
```

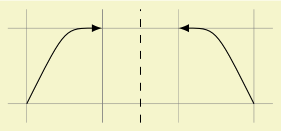
7.2 Using built-in transformations

The following commands use a combination of shifting, rotating, −1 scaling, rotating back and shifting back to reach the mirror transformation.

The commands are named the same as above, only the `m` in `mirror` is capitalized.

`\pgftransformxMirror{`⟨*value*⟩`}`

Sets up a transformation that mirrors along a vertical line that goes through point (⟨*value*⟩, 0).



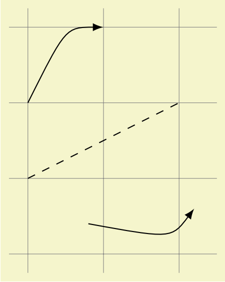
```
\usepgflibrary {transformations.mirror}
\begin{tikzpicture}
\draw[help lines] (-0.25, -.25) grid (3.25, 1.25);
\draw[-latex] (0,0) .. controls (.5,1) .. (1,1);

\draw[dashed] (1.5, -.25) -- (1.5, 1.25);
\pgftransformxMirror{1.5}

\draw[-latex] (0,0) .. controls (.5,1) .. (1,1);
\end{tikzpicture}
```

`\pgftransformyMirror{`⟨*value*⟩`}`

Sets up a transformation that mirrors along a horizontal line that goes through point (0, ⟨*value*⟩).

`\pgftransformMirror{`⟨*point A*⟩`}{`⟨*point B*⟩`}`

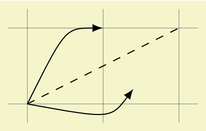
Sets up a transformation that mirrors along the line that goes through ⟨*point A*⟩ and ⟨*point B*⟩.



```
\usepgflibrary {transformations.mirror}
\begin{tikzpicture}
\draw[help lines] (-.25, -2.25) grid (2.5, 1.25);
\draw[-latex] (0,0) .. controls (.5,1) .. (1,1);

\draw[dashed] (0, -1) -- (2, 0);
\pgftransformMirror{\pgfpointxy{0}{-1}}{\pgfpointxy{2}{0}}

\draw[-latex] (0,0) .. controls (.5,1) .. (1,1);
\end{tikzpicture}
```

`\pgfqtransformMirror{`⟨*point A*⟩`}`

Sets up a transformation that mirrors along the line that goes through the origin and ⟨*point A*⟩.



```
\usepgflibrary {transformations.mirror}
\begin{tikzpicture}
\draw[help lines] (-.25, -.25) grid (2.25, 1.25);
\draw[-latex] (0,0) .. controls (.5,1) .. (1,1);

\draw[dashed] (0, 0) -- (2, 1);
\pgfqtransformMirror{\pgfpointxy{2}{1}}

\draw[-latex] (0,0) .. controls (.5,1) .. (1,1);
\end{tikzpicture}
```

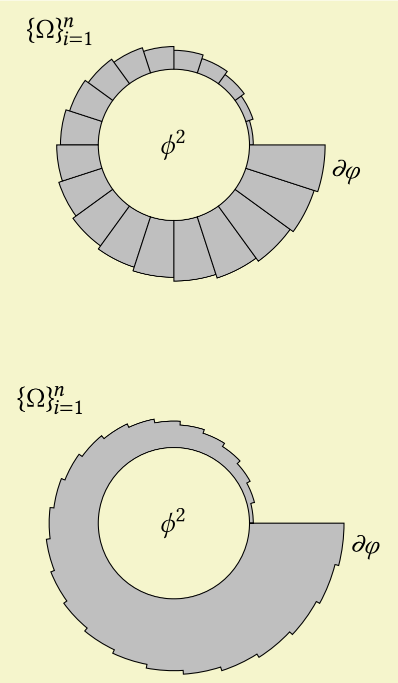
Part IV
Miscellaneous

Ti*k*Z Library misc

```
\usetikzlibrary{misc} % LaTeX and plain TeX
\usetikzlibrary[misc] % ConTeXt
```

This library adds miscelleaneos utilities to PGFmath, PGF or Ti*k*Z.

8 PGFmath



```
\usetikzlibrary {misc}
\begin{tikzpicture}[
  declare function={bigR(\n)=smallR+.05*\n;},
  declare constant={smallR=1; segments=20;},
  full arc=segments]
\foreach \iN[evaluate={\endRadius=bigR(\iN+1);}, use int=0 to segments-1]
  \filldraw[fill=gray!50] (\iN R:\endRadius)
    arc [radius=\endRadius, start angle=\iN R, delta angle=+1R] -- (\iN R+1R:smallR)
    arc [radius=smallR,       end angle=\iN R, delta angle=-1R] -- cycle;

\node                                              {$\phi^2$};
\node at (north west:{sqrt 2 * bigR(segments/2)})  {$\{\Omega\}_{i=1}^n$};
\node[rotate=-.5R, right] at (-.5R: bigR segments) {$\partial \varphi$};

\tikzset{yshift=-5cm, declare constant={segments=25;}, full arc=segments}
\filldraw[fill=gray!50] (right:smallR)
  \foreach \iN[evaluate={\endRadius=bigR(\iN+1);}, use int=0 to segments-1] {
    -- (\iN R:\endRadius) arc[radius=\endRadius, start angle=\iN R, delta angle=1R]}
    -- (right:smallR)     arc[radius=smallR,     start angle=0,     delta angle=-360];

\node                                              {$\phi^2$};
\node at (north west:{sqrt 2 * bigR(segments/2)})  {$\{\Omega\}_{i=1}^n$};
\node[rotate=-.5R, right] at (-.5R: bigR segments) {$\partial \varphi$};
\end{tikzpicture}
```

8.1 Postfix operator R

Similar to `\segments[<num>]` in PSTricks, the postfix operator `R` allows the user to use an arbitrary number of segments of a circle to be used instead of an angle.

`/tikz/full arc=`⟨*num*⟩ (default)

The number ⟨*num*⟩ of segments will be set up. Using `full arc` with an empty value disables the segmentation and `1R` equals 1°.

The given value ⟨*num*⟩ is evaluated when the key is used and doesn't change when ⟨*num*⟩ contains variables that change.

The `R` operator can then be used.

x`R` (postfix operator; uses the `full arc` function)

Multiplies x with $\frac{360}{\langle num \rangle}$.

8.2 Functions

`strrepeat("`*Text*`", `*x*`)`
`\pgfmathstrrepeat{"`*Text*`"}{`*x*`}`

Returns a string with *Text* repeated x times.

foofoofoofoofoo

```
\pgfmathparse{strrepeat("foo", 5)} \pgfmathresult
```

`isInString("`*String*`", "`*Text*`")`
`\pgfmathisInString{"`*String*`"}{"`*Text*`"}`

Returns `1` (true) if *Text* contains *String*, otherwise `0` (false).

0 and 1

```
\pgfmathparse{isInString("foo", "bar")} \pgfmathresult
\ and\
\pgfmathparse{isInString("foo", "foobar")} \pgfmathresult
```

`strcat("`*Text A*`", "`*Text B*`", `…`)`
`\pgfmathstrcat{"`*Text A*`"}{"`*Text B*`"}{`…`}`

Returns the concatenation of all given parameters.

blue!21!green

```
\pgfmathparse{strcat("blue!", int(7*3), "!green")} \pgfmathresult
```

`isEmpty("`*Text*`")`

\pgfmathisEmpty{"*Text*"}

Returns 1 (true) if *Text* is empty, otherwise 0 (false).

0 and 1 and 1

```
\pgfmathparse{isEmpty("foo")} \pgfmathresult\ and\
\pgfmathparse{isEmpty("")}    \pgfmathresult\ and\
\def\emptyText{}
\pgfmathparse{isEmpty("\emptyText")} \pgfmathresult
```

8.3 Functions: using coordinates

The following functions can only be used with PGF and/or TikZ. Since the arguments are usually plain text (and not numbers) one has to wrap them in ".

anglebetween("*p1*", "*p2*")
\pgfmathanglebetween{"*p1*"}{"*p2*"}

Return the angle between the centers of the nodes *p1* and *p2*.

qanglebetween("*p*")
\pgfmathqanglebetween{"*p*"}

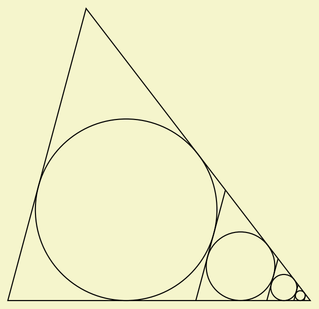
Return the angle between the origin and the center of the node *p*.

distancebetween("*p1*", "*p2*")
\pgfmathdistancebetween{"*p1*"}{"*p2*"}

Return the distance (in pt) between the centers of the nodes *p1* and *p2*.

qdistancebetween("*p*")
\pgfmathqdistancebetween{"*p*"}

Return the distance (in pt) between the origin and the center of the node *p*.



```
\usetikzlibrary {calc,misc,through}
\begin{tikzpicture}
\path (0,0) coordinate (A) + (0:4) coordinate (B) +(75:4) coordinate (C);
\draw (A) -- (B) -- (C) -- cycle;
\foreach \cnt in {1,...,4}{
  \pgfmathsetmacro\triA{distancebetween("B","C")}
  \pgfmathsetmacro\triB{distancebetween("C","A")}
  \pgfmathsetmacro\triC{distancebetween("A","B")}
  \path (barycentric cs:A=\triA,B=\triB,C=\triC) coordinate (M)
       node [draw, circle through=($(A)!(M)!(C)$)] (M) {};
  \draw ($(C)-(A)$) coordinate (vecB)
      (M.75-90) coordinate (@)
      (intersection of @--[shift=(vecB)]@ and B--C) coordinate (C) --
      (intersection of @--[shift=(vecB)]@ and B--A) coordinate (A);}
\end{tikzpicture}
```

9 PGFkeys

9.1 Conditionals

/utils/if=⟨*cond*⟩⟨*true*⟩⟨*false*⟩ (no default)

This key checks the conditional ⟨*cond*⟩ and applies the styles ⟨*true*⟩ if ⟨*cond*⟩ is true, otherwise ⟨*false*⟩. ⟨*cond*⟩ can be anything that PGFmath understands.

As a side effect on how PGFkeys parses argument, the ⟨*false*⟩ argument is actually optional.

The following keys use TEX' macros `\if`, `\ifx`, `\ifnum` and `\ifdim` for faster executions.

/utils/TeX/if=⟨*token A*⟩⟨*token B*⟩⟨*true*⟩⟨*false*⟩ (no default)

This key checks via `\if` if ⟨*token A*⟩ matches ⟨*token B*⟩ and applies the styles ⟨*true*⟩ if it does, otherwise ⟨*false*⟩.

As a side effect on how PGFkeys parses argument, the ⟨*false*⟩ argument is actually optional.

/utils/TeX/ifx=⟨*token A*⟩⟨*token B*⟩⟨*true*⟩⟨*false*⟩ (no default)

As above.

/utils/TeX/ifnum=⟨*num cond*⟩⟨*true*⟩

opt⟨*false*⟩ (no default)

This key checks `\ifnum`⟨*num cond*⟩ and applies the styles ⟨*true*⟩ if true, otherwise ⟨*false*⟩. A delimiting `\relax` will be inserted after ⟨*num cond*⟩.

As a side effect on how PGFkeys parses argument, the ⟨*false*⟩ argument is actually optional.

/utils/TeX/ifdim=⟨*dim cond*⟩⟨*true*⟩⟨*false*⟩ (no default)

As above.

/utils/TeX/ifempty=⟨*Text*⟩⟨*true*⟩⟨*false*⟩ (no default)

This checks whether ⟨*Text*⟩ is empty and applies styles ⟨*true*⟩ if true, otherwise ⟨*false*⟩.

9.2 Handlers

While already a lot of values given to keys are evaluated by PGFmath at some point, not all of them are.

Key handler ⟨*key*⟩/.pgfmath=⟨*eval*⟩

This handler evaluates ⟨*eval*⟩ before it is handed to the key.

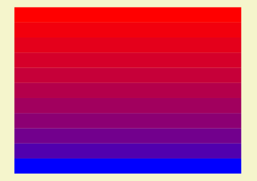
Key handler ⟨*key*⟩/.pgfmath int=⟨*eval*⟩

As above but truncates the result.

Key handler ⟨*key*⟩/.pgfmath strcat=⟨*eval*⟩

As above but uses the `strcat` function.

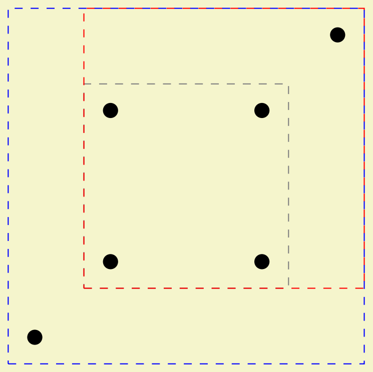
In the example below, one could have used the `/pgf/foreach/evaluate` key from `\foreach`.



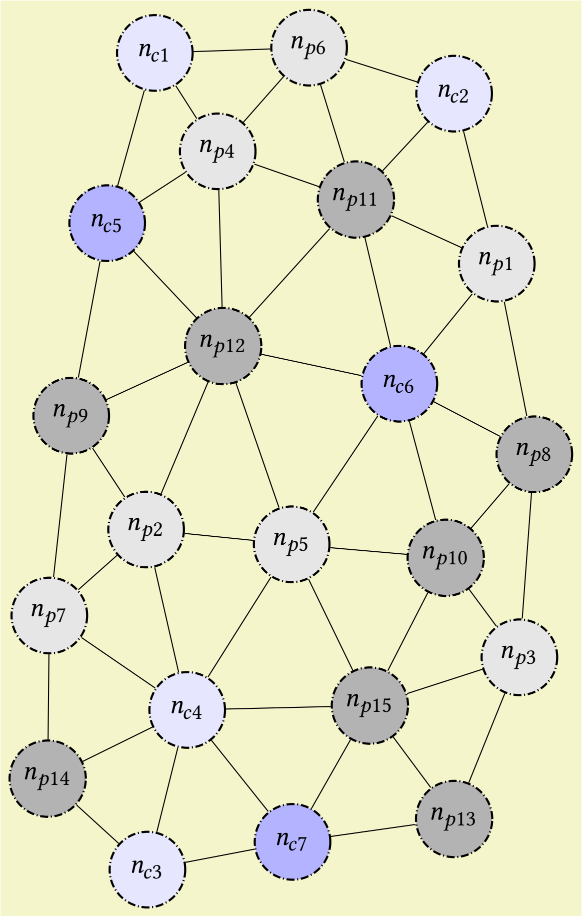
```
\usetikzlibrary {misc}
\tikz\foreach \i in {0,10,...,100}
  \draw[line width=+.2cm, color/.pgfmath strcat={"red!",sqrt(\i)*10,"!blue"}]
    (0,\i/50) -- +(right:3);
```

Key handler ⟨*key*⟩/.List=⟨⟨*e1*⟩, ⟨*e2*⟩, ..., ⟨*en*⟩⟩

This handler evaluates the given list with `\foreach` and concatenates the element and the result is then given to the used key.



```
\usetikzlibrary {fit,misc}
\begin{tikzpicture}[nodes={draw, dashed, inner sep=+10pt}]
  \foreach \point [count=\cnt] in {(0,0), (0,2), (2,0), (2,2), (3,3), (-1,-1)}
    \fill \point circle[radius=.1] coordinate (point-\cnt);
  \node[gray, fit/.List={(point-1),(point-...),(point-4)}] {};
  \node[red,  fit/.List={(point-1),(point-...),(point-5)}] {};
  \node[blue, fit/.List={(point-1),(point-...),(point-6)}] {};
\end{tikzpicture}
```



```
\usetikzlibrary {graphs,graphdrawing} \usegdlibrary {force}
\tikzset{
  mynode/.style={
    circle, minimum size=10mm, draw, densely dashdotted, thick,
    decide color/.expand once=#1},
  decide color/.style 2 args={
    /utils/TeX/if=c#1
      {/utils/TeX/ifnum={#2<5}{bluelight}{bluedark}}
      {/utils/TeX/ifnum={#2<8}{light}{dark}}},
  light/.style={fill=gray!20},  bluelight/.style={fill=blue!10},
  dark/.style ={fill=gray!60},  bluedark/.style ={fill=blue!30}}
\tikz\graph[
  spring electrical layout, vertical=c2 to p13,
  node distance=1.5cm, typeset=$n_{\tikzgraphnodetext}$,
  nodes={mynode=\tikzgraphnodetext}] {
  % outer ring
  c2 -- {p1, p11, p6};
    p1 -- {p8, c6, p11};
      p8 -- {p3, p10, c6};
       p3 -- {p13, p15, p10};
         p13 -- {p15, c7};
           c7  -- {c3, c4, p15};
           c3  -- {p14, c4};
           p14 -- {p7, c4};
         p7 -- {p9, p2, c4};
       p9 -- {c5, p12, p2};
     c5 -- {c1, p4, p12};
   c1 -- {p6, p4};
  p6 -- {p11, p4};
  % inner ring
  p11 -- {c6, p12, p4};
  p5 -- {c6 -- {p10, p12}, p10 -- p15, p15 -- c4, c4 -- p2, p2 -- p12, p12 -- p4};
};
```

10 PGFfor

Instead of `\foreach \var in {start, start + delta, ..., end}` one can use `\foreach \var[use int=start to end step delta]`.

`/pgf/foreach/use int`=⟨*start*⟩`to`⟨*end*⟩`step`⟨*delta*⟩ (no default)

The values ⟨*start*⟩, ⟨*end*⟩ and ⟨*delta*⟩ are evaluates by PGFmath at initialization. The part `step` ⟨*delta*⟩ is optional (⟨*delta*⟩ = 1).

`/pgf/foreach/use float`=⟨*start*⟩ o⟨*end*⟩optstep⟨*delta*⟩ (no default)

Same as above, however the results are not truncated.

Index

This index only contains automatically generated entries. A good index should also contain carefully selected keywords. This index is not a good index.