

Some frequently and not so frequently asked questions about R_X

(version 0.7.1)

Gert-Ludwig Ingold

<gert.ingold@physik.uni-augsburg.de>

Contents

1	General aspects of R_X	3
1.1	The name of the game NEW	3
1.2	Where do I get the latest version of R _X ?	3
1.3	How can I determine the version of R _X running on my machine?	3
1.4	Does R _X run under my favorite operating system?	3
1.5	Under which versions of Python will R _X run?	3
1.6	Does R _X provide a GUI to view the produced image?	3
1.7	Where can I get help if my question is not answered in this FAQ?	4
2	Python	4
2.1	What is Python?	4
2.2	Where can I learn more about Python?	4
2.3	What do I need to import in order to use R _X ?	4
2.4	What is a raw string and why should I know about it when using R _X ?	5
3	Plotting of graphs	5
3.1	General aspects	5
3.1.1	How do I generate a graph from data as simply as possible?	5
3.1.2	How do I generate a graph of a function as simply as possible?	5
3.1.3	How can I stack graphs?	6
3.1.4	How can I plot grid data? CHANGED	6
3.1.5	How can I access points in problem coordinates of a graph? NEW	7
3.2	Axis properties	7
3.2.1	How do I specify the tick increment?	7
3.2.2	How do I plot the zero line?	8
3.2.3	How can I add grid lines to a graph? NEW	8
3.3	Data properties	8
3.3.1	How do I choose the symbol? CHANGED	8
3.3.2	How do I choose the color of the symbols? CHANGED	9
3.3.3	How do I choose the line style? CHANGED	9
4	T_EX and L_AT_EX	9
4.1	General aspects	9
4.1.1	What is T _E X/L _A T _E X and why do I need it?	9
4.1.2	I don't know anything about T _E X and L _A T _E X. Where can I read something about it?	9
4.1.3	What is CTAN?	10
4.1.4	Is there support for ConT _E Xt?	10

4.2	\TeX and \LaTeX commands useful for $\text{\R}\text{\X}$	10
4.2.1	How do I get a specific symbol with \TeX or \LaTeX ?	10
4.3	\TeX and \LaTeX errors	10
4.3.1	Undefined control sequence <code>\usepackage</code>	10
4.3.2	Undefined control sequence <code>\frac</code>	10
4.3.3	Missing \$ inserted	10
4.3.4	Why do environments like <code>itemize</code> or <code>eqnarray</code> seem not to work? NEW	11
4.3.5	Font shape ‘OT1/xyz/m/n’ undefined	11
4.3.6	File . . . is not available or not readable	11
4.3.7	No information for font ‘cmr10’ found in font mapping file CHANGED	11
4.4	Fonts	12
4.4.1	I want to use a font other than computer modern roman	12
4.4.2	Can I use a TrueType font with $\text{\R}\text{\X}$?	13

Acknowledgements

The following persons have in one way or the other, e.g. by asking good questions or providing answers, contributed to this FAQ:

Walter Briskens, Jörg Lehmann, Michael Schindler, André Wobst.

1 General aspects of $\text{\texttt{PyX}}$

1.1 The name of the game NEW

Originally, the name $\text{\texttt{PyX}}$ was constructed as a combination of **P**ostscript, i.e. the first output format supported by $\text{\texttt{PyX}}$, **P**ython, i.e. the language in which $\text{\texttt{PyX}}$ is written, and **T**e**X**, i.e. the program which $\text{\texttt{PyX}}$ uses for typesetting purposes. Actually, the title of this question is a tribute to $\text{\texttt{T}}\text{\texttt{E}}\text{\texttt{X}}$ because it is taken from the first chapter of the $\text{\texttt{T}}\text{\texttt{E}}\text{\texttt{X}}$ book¹ where the origin of the name $\text{\texttt{T}}\text{\texttt{E}}\text{\texttt{X}}$ and its pronunciation are explained.

Despite the ties between $\text{\texttt{T}}\text{\texttt{E}}\text{\texttt{X}}$ and $\text{\texttt{PyX}}$, their pronunciation is quite different. According to the developers of $\text{\texttt{PyX}}$, it should be pronounced as [pyks]. Please do not pronounce it as [pyx] or [pyç].

1.2 Where do I get the latest version of $\text{\texttt{PyX}}$?

The current release of $\text{\texttt{PyX}}$ (as well as older ones) is freely available from <http://pyx.sourceforge.net> where also a CVS repository with the latest patches can be found. Possibly older versions of $\text{\texttt{PyX}}$ are also available as package for various Linux distributions: see, for instance, <http://packages.debian.org/testing/python/python-pyx.html> for information on the $\text{\texttt{PyX}}$ package in Debian GNU/Linux, <http://packages.gentoo.org/ebuilds/?pyx-0.3.1> for a Gentoo Linux ebuild, and http://www.suse.de/en/private/products/suse_linux/i386/packages_professional/python-pyx.html for the $\text{\texttt{PyX}}$ package in the SUSE LINUX professional distribution.

1.3 How can I determine the version of $\text{\texttt{PyX}}$ running on my machine?

Start a python session (usually by typing `python` at the system prompt) and then type the following two commands (`>>>` is the python prompt)

```
>>> import pyx
>>> pyx.__version__
```

1.4 Does $\text{\texttt{PyX}}$ run under my favorite operating system?

Yes, if you have installed Python ([↑2.1](#)) and $\text{\texttt{T}}\text{\texttt{E}}\text{\texttt{X}}$ ([↑4.1.1](#)). Both are available for a large variety of operating systems so chances are pretty good that you will get $\text{\texttt{PyX}}$ to work on your system.

1.5 Under which versions of Python will $\text{\texttt{PyX}}$ run?

$\text{\texttt{PyX}}$ is supposed to work with Python 2.0 and above. However, most of the development takes place under the current production version of Python (2.3.3 by the time of this writing) and thus $\text{\texttt{PyX}}$ is better tested with this version. On the other hand, the examples and tests are verified to run with all Python versions 2.x. $\text{\texttt{PyX}}$ will not work with Python 1.x due to missing language features.

The version of your Python interpreter can be determined by calling it with the option `-V`. Alternatively, you can simply start the interpreter and take a look at the startup message. Note that there may be different versions of Python installed on your system at the same time. The default Python version need not be the same for all users.

1.6 Does $\text{\texttt{PyX}}$ provide a GUI to view the produced image?

No, $\text{\texttt{PyX}}$ itself does not provide a means to view the produced image. The result of a $\text{\texttt{PyX}}$ run is an EPS (= Encapsulated PostScript) file which can be viewed, printed or imported into other applications. There are several means of viewing EPS files. A common way would be to use `ghostview` which provides a user interface to the PostScript interpreter `ghostscript`. More information about this software, which is

¹D. Knuth, *The $\text{\texttt{T}}\text{\texttt{E}}\text{\texttt{X}}$ book* (Addison-Wesley, 1984)

available for a variety of platforms, can be found at <http://www.cs.wisc.edu/~ghost/>. If you do not own a printer which is capable of printing PostScript files directly, ghostscript may also be useful to translate the EPS file produced by R_X into something your printer will understand.

1.7 Where can I get help if my question is not answered in this FAQ?

The R_X sources contain a reference manual which is also available online at <http://pyx.sourceforge.net/manual/>. Furthermore, there exists a set of examples demonstrating various features of R_X, which is available in the sources or can be browsed at <http://pyx.sourceforge.net/examples.html>. If the feature you are looking for is among them, using the appropriate part of the example code or adapting it for your purposes may help.

There is also a user discussion list about R_X which you can subscribe to at <http://lists.sourceforge.net/lists/listinfo/pyx-user>. The archive of the discussion list is available at http://sourceforge.net/mailarchive/forum.php?forum_id=23700.

Finally, it might be worth checking <http://pyx.sourceforge.net/pyxfaq.pdf> for an updated version of this FAQ.

2 Python

2.1 What is Python?

From www.python.org:

Python is an *interpreted, interactive, object-oriented* programming language. It is often compared to Tcl, Perl, Scheme or Java.

Python combines remarkable power with very clear syntax. It has modules, classes, exceptions, very high level dynamic data types, and dynamic typing. There are interfaces to many system calls and libraries, as well as to various windowing systems (X11, Motif, Tk, Mac, MFC). New built-in modules are easily written in C or C++. Python is also usable as an extension language for applications that need a programmable interface.

The Python implementation is portable: it runs on many brands of UNIX, on Windows, OS/2, Mac, Amiga, and many other platforms. If your favorite system isn't listed here, it may still be supported, if there's a C compiler for it. Ask around on [news:comp.lang.python](http://news.comp.lang.python) — or just try compiling Python yourself.

The Python implementation is **copyrighted** but **freely usable and distributable, even for commercial use**.

2.2 Where can I learn more about Python?

The place to start is www.python.org where you will find plenty of information on Python including tutorials.

2.3 What do I need to import in order to use R_X?

It is recommended to begin your Python code with

```
from pyx import *
```

when using R_X. This allows you for example to write simply `graph.graphxy` instead of `pyx.graph.graphxy`. The following modules will be loaded: `attr`, `box`, `canvas`, `color`, `connector`, `data`, `deco`, `epsfile`, `graph`, `path`, `style`, `trafo`, `text`, and `unit`.

For convenience, you might import specific objects of a module like in

```
from graph import graphxy
```

which allows you to write `graphxy()` instead of `graph.graphxy()`.

All code segments in this document assume that the import line mentioned in the first code snippet is present.

2.4 What is a raw string and why should I know about it when using \R X ?

The backslash serves in standard Python strings to start an escape sequence. For example `\n` corresponds to a newline character. On the other hand, \TeX and \LaTeX , which do the typesetting in \R X , use the backslash to indicate the start of a command. In order to avoid the standard interpretation, the string should be marked as a raw string by prepending it by an `r` like in

```
c.text(0, 0, r"$\alpha\beta\gamma$")
```

3 Plotting of graphs

3.1 General aspects

3.1.1 How do I generate a graph from data as simply as possible?

Suppose that you have a data file `x.dat` containing values for x and y in two columns. Then the following code will do the job

```
from pyx import *

g = graph.graphxy(width=10)
g.plot(graph.data.file("x.dat", x=1, y=2))
g.writeEPSfile("x")
```

`graphxy` creates a canvas (called `g` in this example) onto which the graph will be drawn and it sets the default behavior including the axis. There is, however, no default value for the width of the graph. In `plot` you have to specify the name of the data file and the columns from which the data should be taken. Finally, `writeEPSfile` will generate the postscript file `x.eps` which you can view or print.

A minimal example is also provided in the \R X distribution as `examples/graphs/minimal.py`.

3.1.2 How do I generate a graph of a function as simply as possible?

The following example will draw a parabola:

```
from pyx import *

g = graph.graphxy(width=10,
                  x=graph.axis.linear(min=-2, max=2)
                  )

g.plot(graph.data.function("y=x**2"))

g.writeEPSfile("x")
```

Most of the code has been explained in [↑3.1.1](#). The main difference is that here you need to specify minimum and maximum for the x -axis so that \R X knows in which range to evaluate the function.

Another, slightly more complex, example is also provided in the \R X distribution as `examples/graphs/piaxis.py`.

3.1.3 How can I stack graphs?

R_X always needs a canvas to draw on. One possibility therefore consists in creating a new canvas with

```
c = canvas.canvas()
```

and inserting the graphs into this canvas with `c.insert(...)`. Here, `...` has to be replaced by the name of the graph. Alternatively, the canvas created with `graph.graphxy` for one of the graphs can be used to insert the other graphs even if they will be positioned outside the first graph.

The second issue to address is positioning of the graphs. By specifying `xpos` and `ypos` when calling `graphxy`, you can define the position of a graph. Later on, the position and size of a graph `g` can be referred to as `g.xpos`, `g.ypos`, `g.width`, and `g.height` even if for example the height has never been specified explicitly but is only defined by a R_X default.

The following example shows how to put graph `gupper` above graph `glower` on a canvas `c`:

```
from pyx import *
from graph import graphxy

c = canvas.canvas()

glower = graphxy(width=10)
glower.plot(...)
c.insert(glower)

gupper = graphxy(width=10, ypos=glower.ypos+glower.height+2)
gupper.plot(...)

c.insert(gupper)
c.writeEPSfile(...)
```

where `...` has to be replaced by the appropriate information like data and symbol specifications and the name of the output file. Here, `c.insert` is used to actually insert the subcanvasses for the graphs into the main canvas `c` and `c.writeEPSfile` in the last line requests to write the contents of this canvas to a file.

3.1.4 How can I plot grid data?

CHANGED

R_X offers support for plotting three-dimensional data as two-dimensional color plots or grey-scale plots and of vector fields by providing ways to plot rectangles and arrows in graphs.

We start by considering the task of creating a two-dimensional color plot by plotting a number of filled rectangles. One first needs to create a data set which consists of five entries per data point. These are the lower left corner (x_{\min}, y_{\min}) and the upper right corner (x_{\max}, y_{\max}) of the rectangle and a value between 0 and 1 determining the color via a R_X color palette. The following code gives an idea of how to proceed:

```
g.plot(graph.data.file("datafile.dat"), xmin=1, xmax=2, ymin=3, ymax=4, color=5),
      [graph.style.rect(color.palette.ReverseRainbow)]
)
g.dodata()
```

Here, we assume that the data are stored in `datafile.dat` and the columns contain x_{\min} , x_{\max} , y_{\min} , y_{\max} , and the color value in this order. The columns are numbered from 1, since the 0th column contains the line number. To determine the color, we use the `ReverseRainbow` palette. The last line instructs R_X to plot the rectangles before plotting the axes. Otherwise, the axes might be covered partially by the rectangles and, in particular, the ticks might not be visible. Gray-scale plots can easily be generated by specifying the palette `Gray` or `ReverseGray` (cf. appendix C of the manual for a list of predefined palettes).

At first sight, it seems surprising that plotting of grid data requires the specification of four coordinates for the rectangle. The reason is that this allows to draw rectangles of varying sizes which may help to reduce the size of the postscript file by combining rectangles of the same color in horizontal or vertical direction. For example, it may be sufficient to plot a grey-scale image in a small number of grey shades and then combining rectangles may be appropriate. Note, though, that this step is part of the data creation and not preformed by `RyX`. Another advantage of fully specifying each rectangle is that it is straightforward to leave parts of the graph blank.

The same ideas as for the color plot can be applied to plot vector fields where each data point is represented by an arrow. In this case a data point is specified by the position of the arrow, its size and its direction as indicated in the following code snippet:

```
g.plot(graph.data.file("datafile.dat"), x=1, y=2, size=3, angle=4),
      [graph.style.arrow()]
    )
```

Complete code examples can be found in `examples/graphs/mandel.py` and `examples/graphs/arrows.py`.

3.1.5 How can I access points in problem coordinates of a graph?

NEW

Sometimes it may be necessary to add graphical elements to a graph in addition to the data or function(s) which have been plotted as described in [↑3.1.1](#) and [↑3.1.2](#). For a graph instance `g` the positioning can easily be done in canvas coordinates by making use of the origin (`g.xpos`, `g.ypos`) and the width (`g.width`) and height (`g.height`) of the graph.

Occasionally, it may be more convenient to specify the position of the additional material in terms of problem coordinates. However, this requires that the mapping from problem coordinates to canvas coordinates is known. By default this is not the case before the content of the canvas is written to the output which is too late for our purpose. One therefore needs to explicitly instruct `RyX` to determine this mapping. One possibility is to ask `RyX` to finish the graph by means of `g.finish()`. Now, problem coordinates can be used to insert additional material which will end up in front of the graph. If this is not desired, one should only fix the layout of the graph by means of `g.dolayout()`. Then, the additional material can be put onto the canvas before the graph is drawn and it will therefore appear behind the graph.

The conversion of problem coordinates (`px`, `py`) to canvas coordinates (`x`, `y`) is performed as follows:

```
x, y = g.pos(px, py)
```

By default, the problem coordinates will refer to the ranges of the x and y axes. If several axes with different ranges exist, the instances of the desired axes should be passed to the `pos` method by means of the keyword arguments `xaxis` and `yaxis`.

We remark that the drawing of lines parallel to one of the axes at specific problem coordinates can also be done by adapting the method described in [↑3.2.2](#).

3.2 Axis properties

3.2.1 How do I specify the tick increment?

In the partition of a linear axis, the increments associated with ticks, subticks etc. can be specified as argument of `parter.linear`. In the following example, ticks will be drawn at even values while subticks will be drawn at all integers:

```
from pyx.graph import axis
tg = graph.graphxy(width=10,
                  x=axis.linear(min=1, max=10,
                                parter=axis.parter.linear(tickdist=[2,1]))
                  )
```

3.2.2 How do I plot the zero line?

RyX releases before 0.6 offered the possibility to stroke a zero line by specifying `zeropathattrs` in the painter constructor. In more recent releases, one proceeds as follows. First one has to fix the layout information of the graph by means of the `finish` or `doLayout` method (see 3.1.5 for a more detailed explanation). Then, the `xgridpath` or `ygridpath` method of a graph will return a grid path parallel to the y or x axis, respectively, at the specified y value. As an example, a zero line in x direction can be drawn as follows:

```
g.finish()
g.stroke(g.ygridpath(0))
```

3.2.3 How can I add grid lines to a graph? NEW

Specifying `gridattrs` for the painter of an axis will generate grid lines orthogonal to this axis. At least an empty list is needed like in

```
g = graph.graphxy(width=10,
                  x=graph.axis.linear(painter=graph.axis.painter.regular(gridattrs=[])),
                  y=graph.axis.linear()
                  )
```

where grid lines in vertical direction are drawn in default style.

Occasionally, one might want to draw grid lines corresponding to ticks and subticks in a different style. This can be achieved by specifying changeable attributes using `changelist`. The following code

```
my_xpainter = graph.axis.painter.regular(gridattrs=
    [attr.changelist([style.linestyle.solid, style.linestyle.dashed])]
    )
my_ypainter = graph.axis.painter.regular(gridattrs=
    [attr.changelist([color.rgb.red, color.rgb.blue])]
    )

g = graph.graphxy(width=10,
                  x=graph.axis.linear(painter=my_xpainter),
                  y=graph.axis.linear(painter=my_ypainter)
                  )
```

will create vertical solid and dashed grid lines for ticks and subticks, respectively. The horizontal grid lines will be red for ticks and blue for subticks. The changeable attributes are applied in a cyclic manner. Therefore, in this example grid lines at subsubticks would be plotted in the same style as for ticks. If this is not desired, the list of attributes should be extended by an appropriate third style. The keyword `None` will switch off the respective level of grid lines in case you want to draw them only e.g. for ticks but not subticks.

3.3 Data properties

3.3.1 How do I choose the symbol? CHANGED

Suppose a graph called `g` has been initialized, e.g. by using `graph.graphxy`. Then, data and the style of their representation in the graph are defined by calling `g.plot` like in the following example in which filled circles are requested:

```
g.plot(graph.data.file("test.dat"),
    [graph.style.symbol(graph.style.symbol.circle, symbolattrs=[deco.filled])]
    )
```


3.3.2 How do I choose the color of the symbols?

CHANGED

Colors are not properties of the symbol as such and can therefore not be specified in `symbolattrs` directly. The color is rather related to the plotting of the symbol as defined by `deco.stroked` or `deco.filled`. With

```
graph.style.symbol(graph.style.symbol.circle,  
                  symbolattrs=[deco.stroked([color.rgb.red]),  
                                deco.filled([color.rgb.green])]  
                  )
```

you will obtain a circle filled in green with a red borderline.

3.3.3 How do I choose the line style?

CHANGED

If you do not want to use symbols, you can set the line style as in this example

```
g.plot(graph.data.file("test.dat"),  
       [graph.style.line([style.linewidth.Thin])]  
       )
```

where the linewidth is set.

If you also want to use symbols, you can combine the symbol and the line style as in

```
g.plot(graph.data.file("test.dat"),  
       [graph.style.line(lineattrs=[style.linewidth.Thin,  
                                     style.linestyle.dashed]),  
       graph.style.symbol(graph.style.symbolline.circle,  
                           symbolattrs=[deco.filled]) ] )
```

to plot the symbols on top of a thin, dashed line. You may alter the order of the styles to plot the line on top of the symbols.

4 T_EX and L^AT_EX

4.1 General aspects

4.1.1 What is T_EX/L^AT_EX and why do I need it?

T_EX is a high quality typesetting system developed by Donald E. Knuth which is available for a wide variety of operating systems. L^AT_EX is a macro package originally developed by Leslie Lamport which makes life with T_EX easier, in particular for complex typesetting tasks. The current version of L^AT_EX is referred to as L^AT_EX 2_ε and offers e.g. improved font selection as compared to the older L^AT_EX 2.09 which should no longer be used.

All typesetting tasks in R_X are finally handed over to T_EX (which is the default) or L^AT_EX, so that R_X cannot do without it. On the other hand, the capabilities of T_EX and L^AT_EX can be used for complex tasks where both graphics and typesetting are needed.

4.1.2 I don't know anything about T_EX and L^AT_EX. Where can I read something about it?

Take a look at CTAN ([↑4.1.3](#)) where in [CTAN:info](#) you may be able to find some useful information. There exists for example “A Gentle Introduction to T_EX” by M. Doob ([CTAN:gentle/gentle.pdf](#)) and “The Not So Short Introduction to L^AT_EX 2_ε” ([CTAN:info/lshort/english/lshort.pdf](#)) by T. Oetiker et al. The latter has been translated into a variety of languages among them korean (which you will not be able to read unless you have appropriate fonts installed) and mongolian.

Of course, it is likely that these documents will go way beyond what you will need for generating graphics with $\text{\R}\text{\X}$ so you don't have to read all of it (unless you want to use $\text{\T}\text{\E}\text{\X}$ or $\text{\L}\text{\A}\text{\T}\text{\E}\text{\X}$ for typesetting which can be highly recommended).

There exists also a number of FAQs on $\text{\T}\text{\E}\text{\X}$ at [CTAN:help](#).

4.1.3 What is CTAN?

CTAN is the Comprehensive TeX Archive Network where you will find almost everything related to $\text{\T}\text{\E}\text{\X}$ and friends. The main CTAN servers are [tug.ctan.org](#), [dante.ctan.org](#), and [cam.ctan.org](#). A list of FTP mirrors can be found at [CTAN:CTAN.sites](#).

In this FAQ, CTAN: refers to the root of an anonymous ftp CTAN tree, e.g. [ftp://ctan.tug.org/tex-archive/](#), [ftp://ftp.dante.de/tex-archive/](#), and [ftp://ftp.tex.ac.uk/tex-archive/](#). The links to CTAN in this document point to one of these servers but you might consider using a FTP mirror closer to you in order to reduce traffic load.

4.1.4 Is there support for Con $\text{\T}\text{\E}\text{\X}$ t?

No, and as far as I know there no plans to provide it in the near future. Given the close ties between Con $\text{\T}\text{\E}\text{\X}$ t and MetaPost, Con $\text{\T}\text{\E}\text{\X}$ t users probably prefer to stick with the latter anyway.

4.2 $\text{\T}\text{\E}\text{\X}$ and $\text{\L}\text{\A}\text{\T}\text{\E}\text{\X}$ commands useful for $\text{\R}\text{\X}$

4.2.1 How do I get a specific symbol with $\text{\T}\text{\E}\text{\X}$ or $\text{\L}\text{\A}\text{\T}\text{\E}\text{\X}$?

A list of mathematical symbols together with the appropriate command name can be found at [CTAN:info/symbols/math/symbols.ps](#). A comprehensive list containing more than 2500 symbols for use with $\text{\L}\text{\A}\text{\T}\text{\E}\text{\X}$ can be obtained from [CTAN:info/symbols/comprehensive/symbols-a4.pdf](#). In some cases it might be necessary to install fonts or packages available from CTAN ([↑4.1.3](#)).

4.3 $\text{\T}\text{\E}\text{\X}$ and $\text{\L}\text{\A}\text{\T}\text{\E}\text{\X}$ errors

4.3.1 Undefined control sequence `\usepackage`

The command `\usepackage` is specific to $\text{\L}\text{\A}\text{\T}\text{\E}\text{\X}$. Since by default $\text{\R}\text{\X}$ uses $\text{\T}\text{\E}\text{\X}$, you have to specify the correct mode:

```
text.set(mode="latex")
```

4.3.2 Undefined control sequence `\frac`

The command `\frac` is only available in $\text{\L}\text{\A}\text{\T}\text{\E}\text{\X}$. In $\text{\T}\text{\E}\text{\X}$ you should use `{a\over b}` in math mode to produce $\frac{a}{b}$. As an alternative you may ask for the $\text{\L}\text{\A}\text{\T}\text{\E}\text{\X}$ mode as explained in [4.3.1](#).

4.3.3 Missing \$ inserted

You have specified $\text{\T}\text{\E}\text{\X}$ - or $\text{\L}\text{\A}\text{\T}\text{\E}\text{\X}$ -code which is only valid in math mode. Typical examples are greek symbols, sub- and superscripts or fractions.

On the $\text{\R}\text{\X}$ level, you can specify math mode for the whole string by using `text.mathmode` as in

```
c.text(0, 0, r"\alpha", text.mathmode)
```

Keep also in mind that the standard Python interpretation of the backslash as introducing escape sequences needs to be prevented [↑2.4](#).

On the \TeX/L\TeX level you should enclose the commands requiring math mode in $\$$'s. As an example, $\$\alpha_i^j\$$ will produce α_i^j . This allows you to specify math mode also for substrings. There exist other ways to specify math mode in \TeX and \LaTeX which are particularly useful for more complex typesetting tasks. To learn more about it, you should consult the documentation [↑4.1.2](#).

4.3.4 Why do environments like `itemize` or `eqnarray` seem not to work?

NEW

An `itemize` environment might result in a \LaTeX error complaining about a “missing `\item`” or an `eqnarray` might lead to a \LaTeX message “missing `\endgroup` inserted” even though the syntax appears to be correct. The \TeX nicol reason is that in \R\TeX text is typeset in left-right mode (LR mode) which does not allow linebreaks to occur. There are two ways out.

If the text material should go in a box of given width, a `parbox` can be used like in the following example:

```
text.set(mode="latex")
c = canvas.canvas()
w = 2
c.text(0, 0, r"\begin{itemize}\item a\item b\end{itemize}", [text.parbox(w)])
```

Occasionally, one would like to have the box in which the text appears to be as small as possible. Then the `fancybox` package developed by Timothy Van Zandt is useful which provides several environments like `Bitemize` and `Beqnarray` which can be processed in LR mode. The relevant part of the code could look like:

```
text.set(mode="latex")
text.preamble(r"\usepackage{fancybox}")
c = canvas.canvas()
c.text(0, 0, r"\begin{Bitemize}\item a\item b\end{Bitemize}")
```

Other environments provided by the `fancybox` package include `Bcenter`, `Bflushleft`, `Bflushright`, `Benumerate`, and `Bdescription`. For more details, the documentation of the package should be consulted.

4.3.5 Font shape ‘OT1/xyz/m/n’ undefined

You have asked to use font `xyz` which is not available. Make sure that you have this font available in Type1 format, i.e. there should be a file `xyz.pfb` somewhere. If your \TeX system is TDS compliant (TDS= \TeX directory structure, cf. [CTAN:tds/draft-standard/tds/tds.pdf](#)) you should take a look at the subdirectories of `TEXMF/fonts/type1`.

4.3.6 File ... is not available or not readable

Such an error message might already occur when running the example file `hello.py` included in the \R\TeX distribution. Usually, the error occurs due to an overly restrictive `umask` setting applied when unpacking the `tar.gz` sources. This may render the file mentioned in the error message unreadable because the python `distutil` installation package doesn’t change the file permissions back to readable for everyone.

If the file exists, the problem can be solved by changing the permissions to allow read access.

4.3.7 No information for font ‘cmr10’ found in font mapping file

CHANGED

Such an error message can already be encountered by simply running the example file `hello.py` included in the \R\TeX distribution. The likely reason is that the \TeX system does not find the `cmr` fonts in Type1 format. \R\TeX depends on these fonts as it does not work with the traditional `pk` fonts which are stored as bitmaps.

Therefore, the first thing to make sure is that the cmr Type1 fonts are installed. In some T_EX installations, the command `kpsewhich cmr10.pfb` will return the appropriate path if the cmr fonts exist in the binary Type1 format (extension pfb) required by R_X. If the command does not work but the TeX system is TDS compliant ([↑4.3.5](#)), a look should be taken at `TEXMF/fonts/type1/bluesky/cm` where TEXMF is the root of the texmf tree.

If the Type1 fonts do not exist on the system, they may be obtained from the CTAN [↑4.1.3](#) at [CTAN: fonts/cm/ps-type1/bluesky](#). See the README for information about who produced these fonts and why they are freely available.

If the Type1 fonts exist, the next step is to take a look at `psfonts.map`. There may be several files with this name on the system, so it is important to find out which one TeX is actually using. `kpsewhich psfonts.map` might give this information.

The most likely problem is that this file does not contain a line telling TeX what to do if it encounters a request for font cmr10, i.e. the following line may be missing

```
cmr10          CMR10          <cmr10.pfb
```

It is probable that the required lines (in practice, you do not just need cmr10) are found in a file named `psfonts.cmz` which resides in `TEXMF/dvips/bluesky`.

One solution is to instruct R_X to read additional map files like `psfonts.cmz` or `psfonts.amz`. This can be achieved by modifying the appropriate `pyxcrc` file which is either the systemwide `/etc/pyxcrc` or `.pyxcrc` in the user's home directory. Here, the names of the map files to be read by R_X should be appended separated by whitespaces like in the following example:

```
[text]
fontmaps = psfonts.map psfonts.cmz psfonts.amz
```

The same effect can be achieved by inserting the following line into the R_X code:

```
text.set(fontmaps="psfonts.map psfonts.cmz psfonts.amz")
```

Note that the default map (`psfonts.map`) has to be specified explicitly.

An alternative approach consists in modifying the T_EX installation by inserting the contents of the desired map files like `psfonts.cmz` into `psfonts.map`. Probably, `psfonts.map` recommends not to do this by hand. In this case the instructions given in the file should be followed. Otherwise, `psfonts.cmz` should be copied into `psfonts.map` while keeping a backup of the old `psfonts.map` just in case. After these changes, R_X most likely will be happy. When inserting `psfonts.cmz` into `psfonts.map` it may be a good idea to include `psfonts.amz` as well. `psfonts.amz` contains information about some more fonts which might be needed at some point. Making these changes to `psfonts.map` will imply that the T_EX system will use the cmr fonts in Type1 format instead of pk format which is actually not a bad thing, in particular if `latex / dvips / ps2pdf` is used to generate PDF output. With fonts in pk format this will look ugly and using Type1 fonts solves this problem as well. When `pdflatex` is used to create PDF files, Type1 fonts will be used anyway.

4.4 Fonts

4.4.1 I want to use a font other than computer modern roman

As long as you have a font in Type1 format available, this should be no problem (even though it may cost you some time to set up things properly).

In the simplest case, your L^AT_EX system contains everything needed. Including the following line into your code will probably work

```
text.set(mode="latex")
text.preamble(r"\usepackage{mathptmx}")
```

and give you Times as roman font.

If you own one of the more common commercial fonts, take a look at [CTAN:fonts](#) and its subdirectories as well as at the web page <http://home.vr-web.de/was/fonts.html> of Walter Schmidt. It is not unlikely that somebody has already done most of the work for you and created the files needed for the font to work properly with \LaTeX . But remember: we are talking about commercial fonts here, so do not expect to find the fonts themselves for free.

If none of these cases applies, you should spend some time reading manuals about font installation, e.g. [CTAN:macros/latex/doc/fntguide.pdf](#) (of course, I do not expect font wizards to read the last few lines).

4.4.2 Can I use a TrueType font with $\text{\R}\text{\X}$?

Not directly as $\text{\R}\text{\X}$ only knows how to handle Type1 fonts (although it is possible to get \LaTeX to work with TrueType fonts). However, you may use `ttf2pt1` (from <http://ttf2pt1.sourceforge.net>) to convert a TrueType font into a Type1 font which you then install in your \TeX system [↑4.4.1](#). You will loose hinting information in the conversion process but this should not really matter on output devices with not too low resolution.