latextool

Dr. Yihsiang Liow (February 21, 2024)

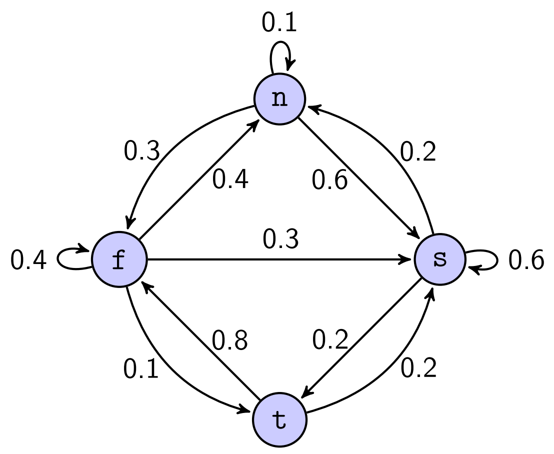
Contents

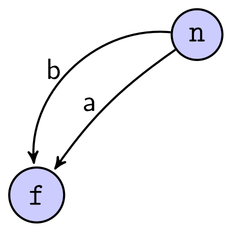
TODO: Shape location: Right now specify x0,y0,x1,y1 or center. Need to add “left= of a”, “left=1cm of a”, “below left=1cm and 3cm of a”,etc. Maybe `{pos = "below 1cm, left 1cm, a"` or `{pos = {'direction':{"below":"1cm", "left":"1cm"}, 'node':'a'}`

TODO: Point, POINT, coordinate are similar. Clean up.

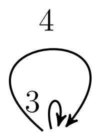
debug: tikz.te

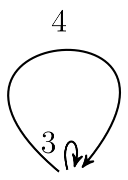
1 tikz stuff





Angle of bend:





debug: tikznc

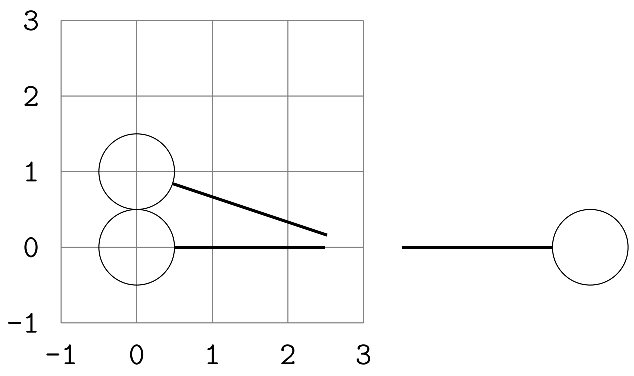
2 Creating tikz node, point, small circles

Something is wrong with the library for drawing very small circles. (Probably error with inner sep: if circle is radius r, innersep should be r/ssqrt(2) – correct and test circle class.) The following is a temporary manual fix using tikz strings.

```
from latextool_basic import *
p = Plot()

p += r"\node[draw,shape=circle,minimum size=0.1cm,fill=blue,inner sep=0](A)
 at (0,0){};"

print(p)
```

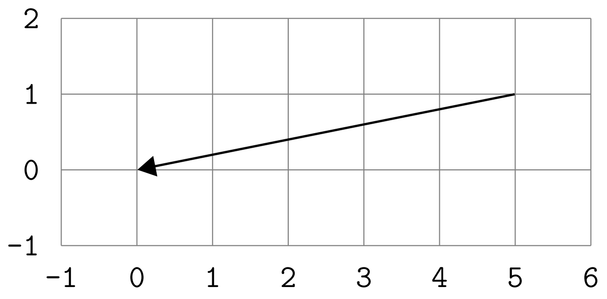
A node with the shape drawn or not drawn

```
from latextool_basic import *
p = Plot()

p += r"\node[draw,shape=circle,minimum size=1cm,inner sep=0](A) at (0,0){};
"
p += r"\node[draw,shape=circle,minimum size=1cm,inner sep=0](D) at (0,1){};
"
p += r"\node[draw=none,shape=circle,minimum size=1cm,inner sep=0](B) at (3,
0){};"
p += r"\node[draw,shape=circle,minimum size=1cm,inner sep=0](C) at (6,0){};
"

p += Line(names=['A', 'B'])
p += Line(names=['D', 'B'])
p += Line(names=['C', 'B'])
p += Grid(-1, -1, 3, 3)
print(p)
```



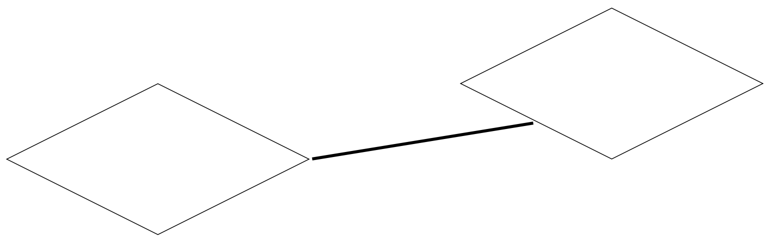
Create tikz node without shape (basically just to have a coordinate with name):

```
from latextool_basic import *
p = Plot()

p += Point(x=0, y=0, name='A')
p += Point(x=5, y=1, name="B")

p += Graph.arc(names=['B','A'])
p += Grid(x0=-1,y0=-1,x1=6,y1=2)
print(p)
```



Creating node wrt to another:

```
from latextool_basic import *
p = Plot()
p += diamond(center=(0, 0), name='a')
p += diamond(center=(6, 1), name='b')

p += r'\node[right=0.0cm of a, inner sep=0.0cm, outer sep=0.0cm] (c) {};'
p += r'\node[below left=0.0cm of b, inner sep=0.0cm, outer sep=0.0cm] (d) {
};'

p += Line(names=['c','d'])
print(p)
```



debug: notes.

3 Notes and API and TODOs

long verbatim:

```
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
```

long console:

```
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
Testing pagebreak
```

ISSUE: The label in the rect is centered. The problem is the font for a character with greater height will appear lower than another. So for an array of rects of character, the base line of the labels are not the same.

ISSUE: The boundary line are drawn within the bounding box. So joining up rects would mean the common line will look thicker. DONE. See Rect2.

ISSUE:

- Joining rects in a uniform way to get hor and ver arrays and 2d arrays. Allows for containers with finite area, infinite horizontally, infinite vertically.
- Where to put first rect? What's the API? For hor array that grows to the right, only need top-left corner or bottom-left.or center-left. Maybe just specify the absolute corners of the firt rect?
- Allow each element of array to be any shape. Use BaseNode of course.

```
CLEAN UP OF PYTHON-LATEX DRAWING LIBRARY

latextool.py
latextool_basic.py

automata

courses/book/graph.py -----------------> projects/latextool/latextool.py
courses/ciss362-automata/resources/dfa/dfa-backup.py
courses/ciss362-automata/resources/dfa/dfa.py
courses/ciss362-automata/resources/dfa/graph/graph.py
projects/automata/dfa/dfa.py

linked list
courses/math325/n/latex_ds.py
projects/test-latex/xxx/latex_ds.py

array sorting

2d graphs:
courses/math325/n/makegraph/plot.py

==============================================================================

automata
```

1d shapes: line or things made up of lines (broom?...) They have

- color
- style
- starting tip: dot, arrow (and arrow style)
- ending tip: dot, arrow (and arrow style)

2d shapes: rect, circle, They have

- boundary is a line. So it has boundary color and boundary style
- interior: color
- minipage

debug: testm

4 Enumerate

4.1 Basic enumerate

For comparison, this is enumerate:

- Line 1.
- line 2.
- line 3.

After enumerate.

This is with paragraphing within items:

- Line 1. This is a test. This is a test. This is a test. This is a test. This is a test. This is a test. This is a test. This is a test.
  Next paragraph. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.
- line 2.
- line 3.

A new paragraph after mylist.

4.2 tightlist

This is the same as mylist except that the vertical spacing before and after the environment is removed. This does not work quite right is the environment is in a paragraph of its own – there will be a little extra vertical spacing before the environment.

Testing tightlist:

- line 1
- line 2
- line 3

After environment.

This is with paragraphing within items:

- Line 1. This is a test. This is a test. This is a test. This is a test. This is a test. This is a test. This is a test. This is a test.
  Next paragraph. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.
- line 2.
- line 3.

A new paragraph after above environment. The next environment is in a new paragraph.

- line 1
- line 2
- line 3

4.3 axioms

The label horizontal spacing is larger.

(GROUP-C) $\forall g, g' \in G$, $gg' \in G$
(GROUP-A) $\forall g, g', g'' \in G$,

debug: textco

5 Text Color

Testing `\textred`: hello world ... $x = 1$ ... $\int x\,dx = \frac{1}{2}x^2 + C$

$$\sin\theta = 0$$

Black again.

```
Testing \verb!\textred!:
\textred{hello world ... $x = 1$ ... $\int x \, dx = \frac{1}{2}x^2 + C$
\[
\sin \theta = 0
\]
}
Black again.
```

Testing `\textblue`: hello world ... $x = 1$ ... $\int x\,dx = \frac{1}{2}x^2 + C$

$$\sin\theta = 0$$

Black again.

Testing `\textwhite`:

Black again.

Using underline/textbox with textwhite:

$$\int x\,dx = \frac{1}{2}x^2 + C$$

$$\int x\,dx = \underline{\hphantom{\frac{1}{2}x^2 + C}}$$

$$\int x\,dx = \boxed{\hphantom{\frac{1}{2}x^2 + C}}$$

debug: pytho

6 Python

There are vertical spacing issues with the python environment. Vertical spacing before and after the python environment has been removed from the python environment in `mypython.tex`.

TEST 1. Python environment with console function call. Line before python environment.

```
hello world
```

Line after the python environment.

TEST 2. One python environment with two console function calls. Line before python environment.

```
hello world 1
```

```
hello world 2
```

Line after the python environment. ISSUE: Note that the two frames are joined up. For the time being do not put two console environment immediately next to each other.

TEST 3. One python environment with two console function calls with one blank line between the console function calls. Line before python environment.

```
hello world 1
```

```
hello world 2
```

Line after the python environment. ISSUE: Note that the two frames are joined up. For the time being do not put two console environment immediately next to each other.

TEST 4. Two python environments with one console function call each. Line before python environment.

```
hello world 1
```

Line between the python environments.

```
hello world 2
```

Line after the second python environment.

TEST 5. This is the console environment in latex (no python).

```
hello world 1
```

Line after.

TEST 6. 1 console environment in latex (no python).

```
hello world 1
```

Line between.

```
hello world 2
```

Line after.

TEST 7. 2 console environments in latex next to each other.

```
hello world 1
```

```
hello world 2
```

Line after.

TEST 8. 1 verbatim environment in latex (no python).

```
hello world 1
```

Line after.

TEST 9. This is 2 Verbatim environments in latex (no python).

```
hello world 1
```

Line between.

```
hello world 2
```

Line after.

TEST 10. 2 Verbatim environments next to each.

```
hello world 1
```

```
hello world 2
```

Line after.

debug: execu

7 execute

The `execute` (in `latexbasic_tool.py`) function executes a python string as a program and the stdout is inserted into the latex file:

```
\begin{python}
from latextool_basic import execute
execute(r"""
print("hello world")
""")
\end{python}
```

Here's the result: hello world

Note that there's some extra whitespace before the hello world. TODO: Need to figure a way to remove the whitespace.

The following does the same except that it first prints the python source in a framed verbatim box:

```
Result:
\begin{python}
from latextool_basic import execute
execute(r"""
print("hello world")
""", print_source=True)
\end{python}
```

If `debug` is set to `True`, there are three outputs: the python source, the stderr, and the stdout, all in framed verbatim boxes.

```
Result:
\begin{python}
from latextool_basic import execute
execute(r"""
print("hello world")
""", debug=True)
\end{python}
```

If there's an error in the python source, the output will be the same as if `debug` is set to `True`.

TEST: debug=True

```
from latextool_basic import execute
execute("print('hello world')", debug=True)
```

Result: PYTHON ERROR. See source, stderr, stdout below

```
print('hello world')
```

```
hello world
```

```
```

TEST: debug=False

```
from latextool_basic import execute
execute("print('hello world')", debug=False)
```

Result: hello world

TEST: print source=True, debug=False

```
from latextool_basic import execute
execute("print('hello world')", print_source=True, debug=False)
```

Result:

```
print('hello world')
```

hello world

TEST: Error in source

```
from latextool_basic import execute
execute("print(hello world)")
```

Result: PYTHON ERROR. See source, stderr, stdout below

```
print(hello world)
```

```

```

```
File "znzddpzs.tmp.py", line 1
    print(hello world)
                    ^

SyntaxError: invalid syntax
```

debug: conso

8 console

NOTE: console function is now the same as the verbatim function.

Here's the spacing between paragraphs. Test test test test test test test test test test test test test test test test test test test test test test test test test test test test test test test.

There a console latex environment and a console function in `latextool_basic`.

TODO: Here's the console environment in latex:

```
hello world
```

Here's the console function call in the python environment:

```
hello world
```

Notice the the second case has a blank line before and after the console window. Fix it.

Here's a python environment that prints hello world, prints a console function return value, prints hello world: hello world

```
hello world
```

hello world

Note that the spacing is different from the above.

The console is just like verbatim:

```
from latextool_basic import console
print(console("hello world"))
```

```
hello world
```

However it wraps around:

```
from latextool_basic import console
print(console(r'''
line 1 ... hello world hello world hello world
line 2 ... hello world hello world hello world hello world hello world hell
o world hello world hello world hello world hello world hello world hello w
orld hello world hello world hello world hello world hello world hello worl
d hello world hello world hello world
'''.strip()))
```

```
line 1 ... hello world hello world hello world
line 2 ... hello world hello world hello world hello world hello world hell
o world hello world hello world hello world hello world hello world hello w
orld hello world hello world hello world hello world hello world hello worl
d hello world hello world hello world
```

You can control the window width:

```
from latextool_basic import console
print(console(r'''
line 1 ... hello world hello world hello world
line 2 ... hello world hello world hello world hello world hello world hell
o world hello world hello world hello world hello world hello world hello w
orld hello world hello world hello world hello world hello world hello worl
d hello world hello world hello world
'''.strip(), width=40))
```

```
line 1 ... hello world hello world hello
 world
line 2 ... hello world hello world hello
 world hello world hello world hello wor
ld hello world hello world hello world h
ello world hello world hello world hello
 world hello world hello world hello wor
ld hello world hello world hello world h
ello world hello world
```

You can specify the `wrapmarker` that is used to indicate a line wrap:

```
from latextool_basic import console
print(console(r'''
line 1 ... hello world hello world hello world
line 2 ... hello world hello world hello world hello world hello world hell
o world hello world hello world hello world hello world hello world hello w
orld hello world hello world hello world hello world hello world hello worl
d hello world hello world hello world
'''.strip(), width=40, wrapmarker='---> '))
```

```
line 1 ... hello world hello world hello
--->  world
line 2 ... hello world hello world hello
--->  world hello world hello world hello wor
---> ld hello world hello world hello world h
---> ello world hello world hello world hello
--->  world hello world hello world hello wor
---> ld hello world hello world hello world h
---> ello world hello world
```

TODO: Fix non-pagebreak.

How to insert → symbol into verbatim area?

debug: verba

9 verbatim

The `verbatim` function allows you to create verbatim environments. Executing tex commands in the environment can be easier.

Example:

```
from latextool_basic import verbatim
print(verbatim("hello world\nlorem ipsum"))
```

```
hello world
lorem ipsum
```

You can have no frame:

```
from latextool_basic import verbatim
print(verbatim("hello world\nlorem ipsum", frame=None))
```

```
hello world
lorem ipsum
```

You have line numbers on the left:

```
from latextool_basic import verbatim
print(verbatim("hello world\nlorem ipsum", numbers='left'))
```

```
1  hello world
2  lorem ipsum
```

You have line numbers on the right:

```
from latextool_basic import verbatim
print(verbatim("hello world\nlorem ipsum", numbers='right'))
```

```
hello world    1
lorem ipsum    2
```

You can escape to a tex command:

```
t = r"""
#include <iostream>

int main()
{
    std::cout << "hello world" << std::endl;
    return 0;
}
"""
from latextool_basic import verbatim
print(verbatim(t.strip(), command=['redtext', 'iostream']))
```

```
#include <iostream>

int main()
{
    std::cout << "hello world" << std::endl;
    return 0;
}
```

You can specify a list of words for each tex command:

```
t = r"""
#include <iostream>

int main()
{
    std::cout << "hello world" << std::endl;
    return 0;
}
"""
from latextool_basic import verbatim
print(verbatim(t.strip(), command=['redtext', ['iostream', 'std']]))
```

```
#include <iostream>

int main()
{
    std::cout << "hello world" << std::endl;
    return 0;
}
```

You can use regular expressions:

```
t = r"""
#include <iostream>

int main()
{
    std::cout << "hello world" << std::endl;
    return 0;
}"""
from latextool_basic import verbatim
print(verbatim(t.strip(), command=['redtext', '"[a-z ]*"']))
```

```
#include <iostream>

int main()
{
    std::cout << "hello world" << std::endl;
    return 0;
}
```

WARNING. Since search is by regular expression, be careful to escape metacharacters:

```
from latextool_basic import verbatim
t = r"""
#include <iostream>

int main()
{
    std::cout << "hello world" << std::endl;
    return 0;
}
"""

print(verbatim(t.strip(), command=['textbox', r'main\(\)']))
```

```
#include <iostream>

int main()
{
    std::cout << "hello world" << std::endl;
    return 0;
}
```

You can also specify which part of the text should be escaped by specifying 3 numbers: the line number, the starting column number, and the ending column number.

```
t = r"""
#include <iostream>

int main()
{
    std::cout << "hello world" << std::endl;
    return 0;
}
"""
from latextool_basic import verbatim
print(verbatim(t.strip(), command=['underline', [4, 4, 13]]))
```

```
#include <iostream>

int main()
{
    std::cout << "hello world" << std::endl;
    return 0;
}
```

You can also specify a linenumber and a text:

```
t = r"""
#include <iostream>

int main()
{
    std::cout << "hello world" << std::endl;
    std::cout << "hello world" << std::endl;
    std::cout << "hello world" << std::endl;
    return 0;
}
"""
from latextool_basic import verbatim
print(verbatim(t.strip(), command=['textred', [5, 'std::cout']]))
```

```
#include <iostream>

int main()
{
    std::cout << "hello world" << std::endl;
    std::cout << "hello world" << std::endl;
    std::cout << "hello world" << std::endl;
    return 0;
}
```

You can have a list of tex commands:

```
from latextool_basic import verbatim
print(verbatim("the answer is 42",
               commands=[
                         ['underline', 'the'],
                         ['textbox', ['answer','42']],
                        ]
              ))
```

$\underline{\text{the}}$ $\boxed{\text{answer}}$ `is` $\boxed{42}$

You can also use the contents of a file like this:

```
from latextool_basic import verbatim
print(verbatim(filename='helloworld.cpp'))
```

`verbatim` is not exactly a verbatim function: there's line wrap. By default the width is set to 75. You can set the width to 40:

```
from latextool_basic import verbatim
print(verbatim("abcdeabcdeabcdeabcdeabcdeabcdeabcdeabcdeabcde", width=10))
```

```
abcdeabcde
abcdeabcde
abcdeabcde
abcdeabcde
abcde
```

You can also specify a string to indicate line wrap like this:

```
from latextool_basic import verbatim
print(verbatim("abcdeabcdeabcdeabcdeabcdeabcdeabcdeabcdeabcde", width=10,
               wrapmarker='---> '))
```

```
abcdeabcde
---> abcdeabcde
---> abcdeabcde
---> abcdeabcde
---> abcde
```

TEST 1. Test vertical spacing before and after verbatim(). Before.

```
hello world 1
hello world 2
```

After.

TEST 2. Test vertical spacing between two verbatim() in 1 python environment. Before.

```
frame 1
hello world 1
hello world 2
```

```
frame 2
hello world 1
hello world 2
```

After.

TEST 3. Same as before but print a blank line between the 2 verbatim()s. Before.

```
frame 1
hello world 1
hello world 2
```

```
frame 2
hello world 1
hello world 2
```

After.

TEST 4. Test vertical spacing between 2 python environments, each containing 1 verbatim(), no blank line between the python environments. Before.

```
frame 1
hello world 1
hello world 2
```

```
frame 2
hello world 1
hello world 2
```

After.

TEST 5. For user input in verbatim environment. Before.

```
Please enter x: 42
Please enter y and z: 43 44
Please enter login: jdoe
```

After.

debug: shell.t

10 shell

The following are 3 ways to draw a framed verbatim environment.

METHOD 1. The following is a console latex environment:

```
[student@localhost latextool] pwd
home/student
```

Line after the console environment. Here are two such:

```
[student@localhost latextool] pwd
home/student
```

```
[student@localhost latextool] pwd
home/student
```

METHOD 2. The following is a python environment that prints a shell function call:

```
[student@localhost doc] pwd
/home/student/yliow/Documents/work/projects/latextool/doc
```

Line after python environment with shell function call. Here are two such:

```
[student@localhost doc] pwd
/home/student/yliow/Documents/work/projects/latextool/doc
```

```
[student@localhost doc] pwd
/home/student/yliow/Documents/work/projects/latextool/doc
```

METHOD 3. The following is a python environment that prints an execute function call:

```
[student@localhost doc] pwd
/home/student/yliow/Documents/work/projects/latextool/doc
```

Line after python environment with execute function call. Here are two such:

```
[student@localhost doc] pwd
/home/student/yliow/Documents/work/projects/latextool/doc
```

```
[student@localhost doc] pwd
/home/student/yliow/Documents/work/projects/latextool/doc
```

Notice the vertical spacing before and after the framed window is smaller for 1. The verbatim spacing before and after the framed window for 2 and 3 are the same.

METHOD 4. The following is a python environment that prints an execute function call:

```
from latextool_basic import shell
print(shell('pwd'))
```

```
[student@localhost doc] pwd
/home/student/yliow/Documents/work/projects/latextool/doc
```

Line after python environment with execute function call.

Another example:

```
from latextool_basic import shell
print(shell('ls -la c*.py'))
```

```
[student@localhost doc] ls -la c*.py
-rwxrwxrwx. 1 root root   62 Feb 21 18:25 cobhhaxf.tmp.py
-rwxrwxrwx. 1 root root 1810 Jul 22  2020 crowfoot.py
```

Line after python environment with execute function call.

Execute command in another directory

```
from latextool_basic import shell
print(shell('pwd; ls -la .b*', dir='/home/student'))
```

```
[student@localhost ~] pwd; ls -la .b*
/home/student
-rw-------. 1 student student 8315 Feb 21 18:25 .bash_history
-rw-r--r--. 1 student student   18 Dec  6  2019 .bash_logout
-rw-r--r--. 1 student student  141 Dec  6  2019 .bash_profile
lrwxrwxrwx. 1 root    root      54 Feb 18 20:15 .bashrc -> /home/student/shares/
yliow/Documents/work/unix/.bashrc
-rwx------. 1 student student 2813 Sep 22  2021 .bashrc~
-rwxrwxrwx. 1 student student 3029 Sep 28  2022 .bashrc.backup
lrwxrwxrwx. 1 root    root      54 Feb 16 20:44 .bashrc.old -> /home/student/sha
res/yliow/Documents/work/unix/.bashrc
```

no execution

```
from latextool_basic import shell
print(shell('ls -la', execute=False))
```

```
[student@localhost doc] ls -la
```

Executing with error: PYTHON ERROR. See source, stderr, stdout below

```
print(x)
```

```
```

```
Traceback (most recent call last):
  File "cgcarbgx.tmp.py", line 1, in <module>
    print(x)
NameError: name 'x' is not defined
```

debug: plot.t

11 Plot

All drawing go into a `Plot` object. The following should be in a python environment:

```
from latextool_basic import Plot, Circle
p = Plot()
p += Circle(x=1, y=1, r=1)
print(p)
```

or run in the `execute` function in a python environment:

```
s = r"""
from latextool_basic import Plot, Circle
p = Plot()
p += Circle(x=1, y=1, r=1)
print(p)
"""

from latextool_basic import execute
execute(s)
```

A centered tikzpicture environment is then inserted.

You can also add a pgf/tikz string to plot:

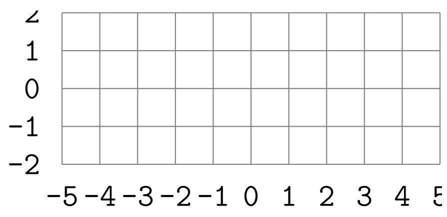
```
from latextool_basic import Plot, Circle
p = Plot()
p += Circle(x=1, y=1, r=1)
p.add("\draw ...")
print(p)
```

You can create and store a pdf image in the `tmp/` subdirectory. See next section.

debug: scalin

12 Scaling

```
from latextool_basic import *
p = Plot(scale=0.5)
p.add(grid, x0=-5, y0=-2, x1=5, y1=2)
print(p)
```



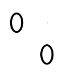
tmp/f28b8b883e26656944de79b8bc132e5f.pdf debug: grid.t

xs

13 Grid

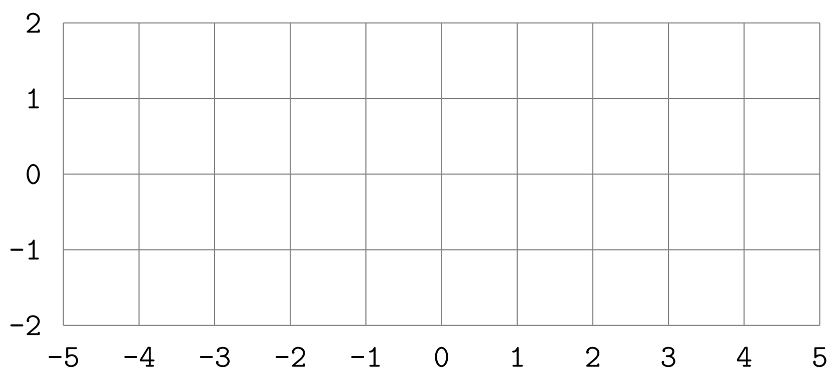
Plot with default grid.

```
from latextool_basic import *
p = Plot()
p += Grid()
print(p)
```



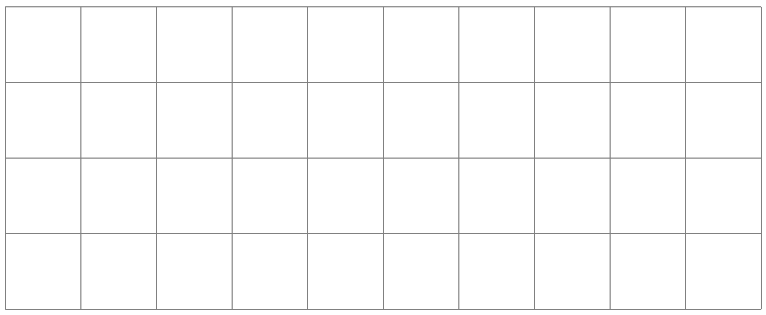
With object grid and bounding box:

```
from latextool_basic import *
p = Plot()
p += Grid(x0=-5, y0=-2, x1=5, y1=2)
print(p)
```



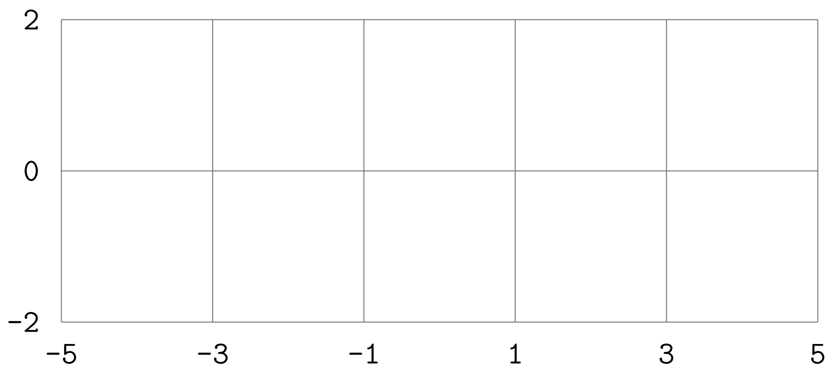
With `label_axis` set to false.

```
from latextool_basic import *
p = Plot()
p += Grid(x0=-5, y0=-2, x1=5, y1=2, label_axes=False)
print(p)
```



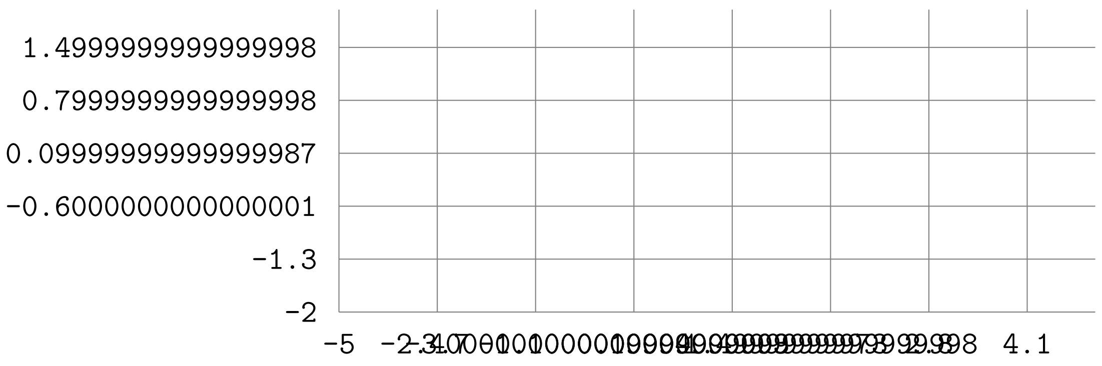
Example. dx = 2, dy = 2.

```
from latextool_basic import *
p = Plot()
p += Grid(x0=-5, y0=-2, x1=5, y1=2, dx=2, dy=2)
print(p)
```



Example. dx = 1.3, dy = 0.7.

```
from latextool_basic import *
p = Plot()
p += Grid(x0=-5, y0=-2, x1=5, y1=2, dx=1.3, dy=0.7)
print(p)
```

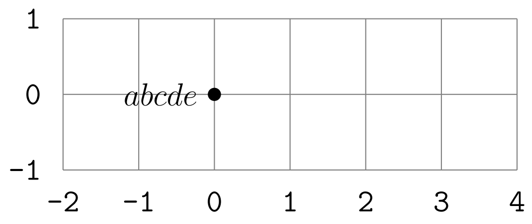

1.4999999999999998
0.7999999999999998
0.09999999999999987
-0.6000000000000001
-1.3
-2
-5
4.1

debug: point.

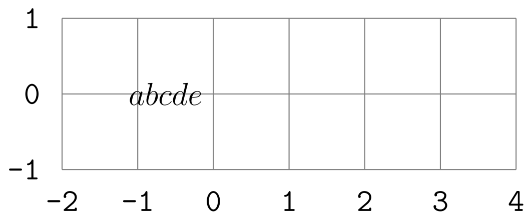
14 Point and text placement

[See section on logic design]

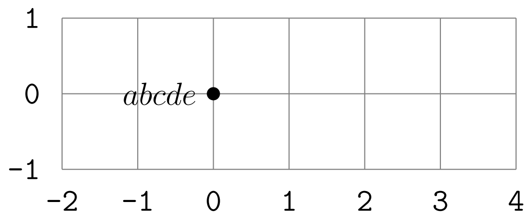
```
from latextool_basic import *
from latexcircuit import *
p = Plot()
X = POINT(x=0, y=0, label='$abcde$')
p += str(X)
p += Grid(x0=-2, y0=-1, x1=4, y1=1)
print(p)
```



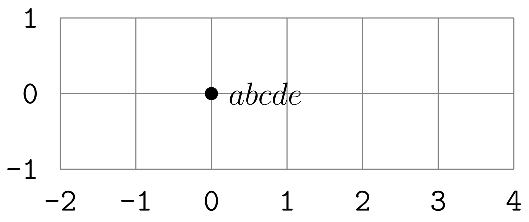
```
from latextool_basic import *
from latexcircuit import *
p = Plot()
X = POINT(x=0, y=0, r=0, label='$abcde$')
p += str(X)
p += Grid(x0=-2, y0=-1, x1=4, y1=1)
print(p)
```



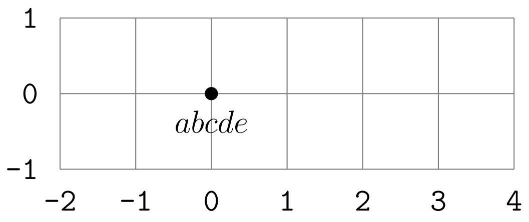
```
from latextool_basic import *
from latexcircuit import *
p = Plot()
X = POINT(x=0, y=0, label='$abcde$', anchor='east')
p += str(X)
p += Grid(x0=-2, y0=-1, x1=4, y1=1)
print(p)
```



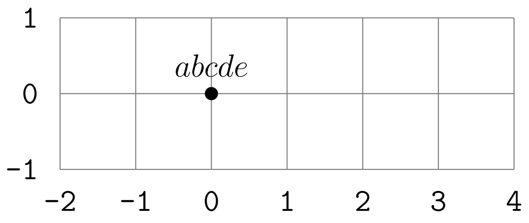
```
from latextool_basic import *
from latexcircuit import *
p = Plot()
X = POINT(x=0, y=0, label='$abcde$', anchor='west')
p += str(X)
p += Grid(x0=-2, y0=-1, x1=4, y1=1)
print(p)
```



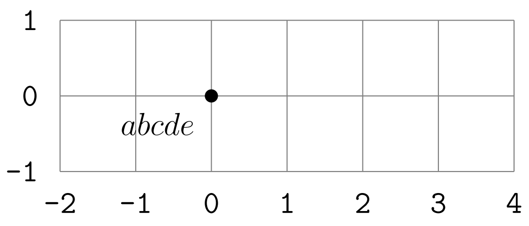
```
from latextool_basic import *
from latexcircuit import *
p = Plot()
X = POINT(x=0, y=0, label='$abcde$', anchor='north')
p += str(X)
p += Grid(x0=-2, y0=-1, x1=4, y1=1)
print(p)
```



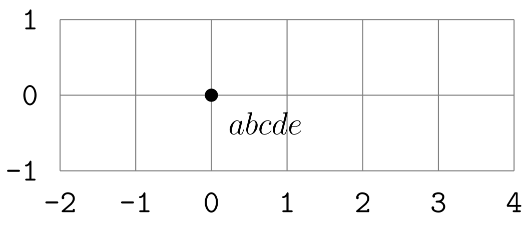
```
from latextool_basic import *
from latexcircuit import *
p = Plot()
X = POINT(x=0, y=0, label='$abcde$', anchor='south')
p += str(X)
p += Grid(x0=-2, y0=-1, x1=4, y1=1)
print(p)
```



```
from latextool_basic import *
from latexcircuit import *
p = Plot()
X = POINT(x=0, y=0, label='$abcde$', anchor='north east')
p += str(X)
p += Grid(x0=-2, y0=-1, x1=4, y1=1)
print(p)
```

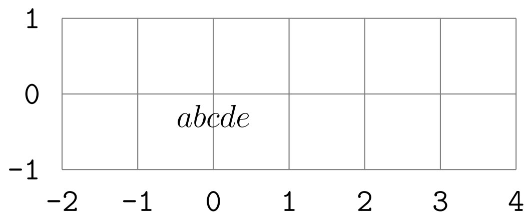


```
from latextool_basic import *
from latexcircuit import *
p = Plot()
X = POINT(x=0, y=0, label='$abcde$', anchor='north west')
p += str(X)
p += Grid(x0=-2, y0=-1, x1=4, y1=1)
print(p)
```

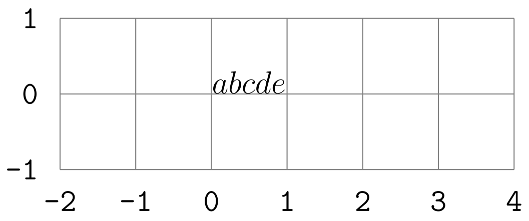


radius 0 cases:

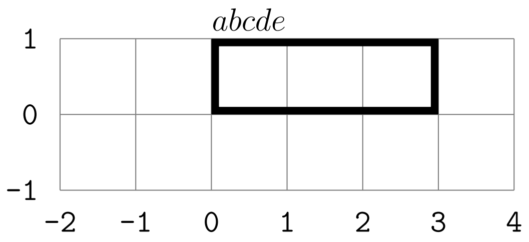
```
from latextool_basic import *
from latexcircuit import *
p = Plot()
X = POINT(x=0, y=0, r=0, label='$abcde$', anchor='north')
p += str(X)
p += Grid(x0=-2, y0=-1, x1=4, y1=1)
print(p)
```



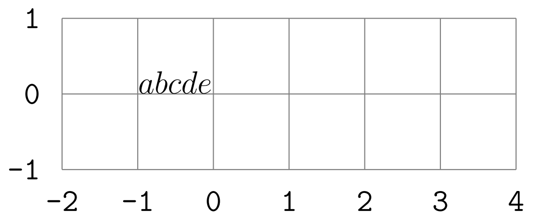
```
from latextool_basic import *
from latexcircuit import *
p = Plot()
X = POINT(x=0, y=0, r=0, label='$abcde$', anchor='flushtopleft')
p += str(X)
p += Grid(x0=-2, y0=-1, x1=4, y1=1)
print(p)
```



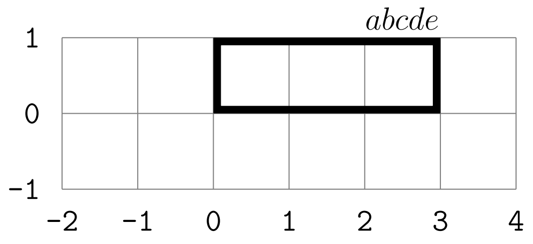
```
from latextool_basic import *
from latexcircuit import *
p = Plot()
p += Rect(x0=0, y0=0, x1=3, y1=1, linewidth=0.1)
X = POINT(x=0, y=1.1, r=0, label='$abcde$', anchor='flushtopleft')
p += str(X)
p += Grid(x0=-2, y0=-1, x1=4, y1=1)
print(p)
```



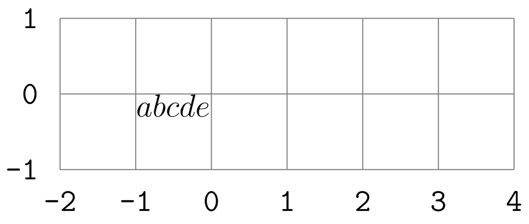
```
from latextool_basic import *
from latexcircuit import *
p = Plot()
X = POINT(x=0, y=0, r=0, label='$abcde$', anchor='flushtopright')
p += str(X)
p += Grid(x0=-2, y0=-1, x1=4, y1=1)
print(p)
```



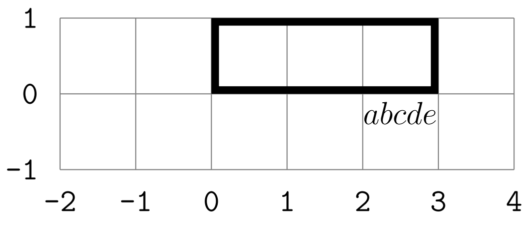
```
from latextool_basic import *
from latexcircuit import *
p = Plot()
p += Rect(x0=0, y0=0, x1=3, y1=1, linewidth=0.1)
X = POINT(x=3, y=1.1, r=0, label='$abcde$', anchor='flushtopright')
p += str(X)
p += Grid(x0=-2, y0=-1, x1=4, y1=1)
print(p)
```



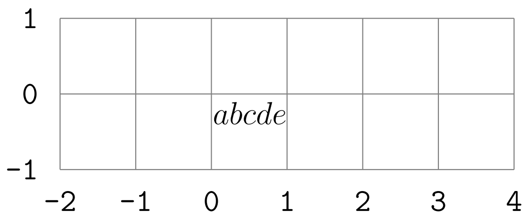
```
from latextool_basic import *
from latexcircuit import *
p = Plot()
X = POINT(x=0, y=0, r=0, label='$abcde$', anchor='flushbottomleft')
p += str(X)
p += Grid(x0=-2, y0=-1, x1=4, y1=1)
print(p)
```



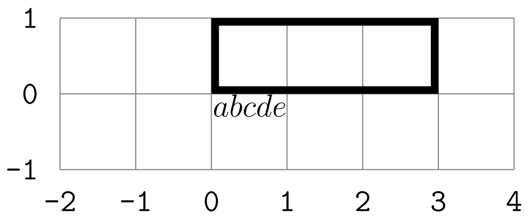
```
from latextool_basic import *
from latexcircuit import *
p = Plot()
p += Rect(x0=0, y0=0, x1=3, y1=1, linewidth=0.1)
X = POINT(x=3, y=-0.1, r=0, label='$abcde$', anchor='flushbottomleft')
p += str(X)
p += Grid(x0=-2, y0=-1, x1=4, y1=1)
print(p)
```



```
from latextool_basic import *
from latexcircuit import *
p = Plot()
X = POINT(x=0, y=-0.1, r=0, label='$abcde$', anchor='flushbottomright')
p += str(X)
p += Grid(x0=-2, y0=-1, x1=4, y1=1)
print(p)
```



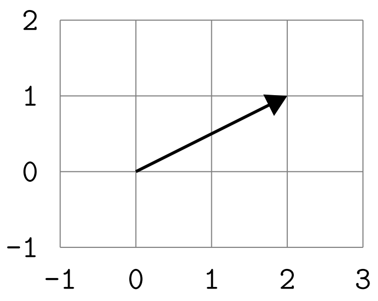
```
from latextool_basic import *
from latexcircuit import *
p = Plot()
p += Rect(x0=0, y0=0, x1=3, y1=1, linewidth=0.1)
X = POINT(x=0, y=0, r=0, label='$abcde$', anchor='flushbottomright')
p += str(X)
p += Grid(x0=-2, y0=-1, x1=4, y1=1)
print(p)
```



debug: coord

15 coordinate

```
from latextool_basic import *
p = Plot()
p += coordinate(0, 0, 'a')
p += coordinate(2, 1, 'b')
p += Line(names=['a', 'b'], endstyle='>')
p += Grid(-1, -1, 3, 2)
print(p)
```



debug: code.t

16 Code

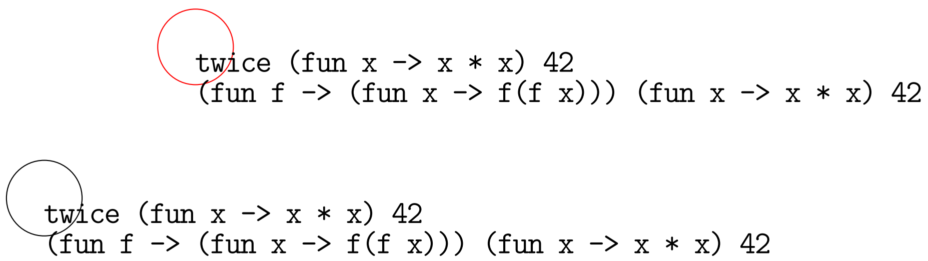
The `code` function creates a table based on the string passed in. The linewidths of the cells are set to 0.

```
from latextool_basic import *
p = Plot()

M = r"""
twice (fun x -> x * x) 42
(fun f -> (fun x -> f(f x))) (fun x -> x * x) 42
""".strip()

code(p, M)

print(p)
```

```
twice (fun x -> x * x) 42
(fun f -> (fun x -> f(f x))) (fun x -> x * x) 42
```

Position:

```
from latextool_basic import *
p = Plot()

M = r"""
twice (fun x -> x * x) 42
(fun f -> (fun x -> f(f x))) (fun x -> x * x) 42
""".strip()

p += Circle(x=0, y=0, r=0.5)
code(p, M, x=0, y=0)

p += Circle(x=2, y=2, r=0.5, linecolor='red')
code(p, M, x=2, y=2)

print(p)
```


twice (fun x -> x * x) 42
(fun f -> (fun x -> f(f x))) (fun x -> x * x) 42
twice (fun x -> x * x) 42
(fun f -> (fun x -> f(f x))) (fun x -> x * x) 42

16.1 Cell size

```
from latextool_basic import *
p = Plot()

M = r"""
twice (fun x -> x * x) 42
(fun f -> (fun x -> f(f x))) (fun x -> x * x) 42
""".strip()

code(p, M, width=0.3, height=1)

print(p)
```

```
twice (fun x -> x * x) 42

(fun f -> (fun x -> f(f x))) (fun x -> x * x) 42
```

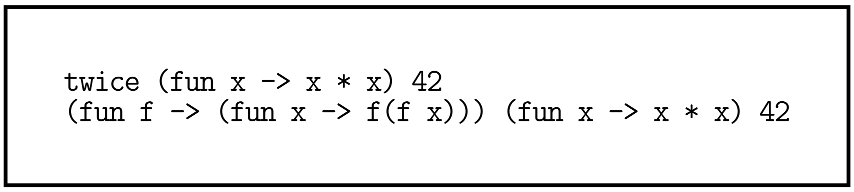
16.2 border linewidth

```
from latextool_basic import *
p = Plot()

M = r"""
twice (fun x -> x * x) 42
(fun f -> (fun x -> f(f x))) (fun x -> x * x) 42
""".strip()

code(p, M, border_linewidth=0.05)

print(p)
```

```
twice (fun x -> x * x) 42
(fun f -> (fun x -> f(f x))) (fun x -> x * x) 42
```

16.3 distance of border to rect

```
from latextool_basic import *
p = Plot()

M = r"""
twice (fun x -> x * x) 42
(fun f -> (fun x -> f(f x))) (fun x -> x * x) 42
""".strip()

code(p, M, border_linewidth=0.05, innersep=0.8)

print(p)
```

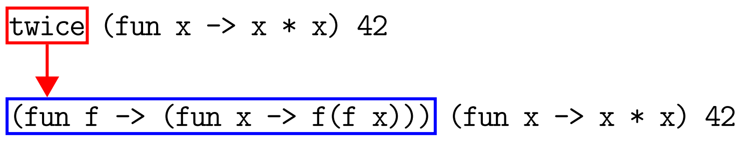


```
twice (fun x -> x * x) 42
(fun f -> (fun x -> f(f x))) (fun x -> x * x) 42
```

16.4 coderect

The return of `code` is just a rect. `coderect` takes in a `code` rect and starting row and column indices and ending row and column indices and returns a rect. The linecolor of this rect is red.

```
from latextool_basic import *
p = Plot()

M = r"""
twice (fun x -> x * x) 42


(fun f -> (fun x -> f(f x))) (fun x -> x * x) 42
""".strip()

N = code(p, M)
r0 = coderect(N, 0, 0, 0, 4)
p += r0
r1 = coderect(N, 3, 0, 3, 27)
r1.linecolor='blue'
p += r1

p0 = r0.bottom()
p1 = r1.top(); p1 = (p0[0], p1[1])
p += Line(points=[p0,p1], endstyle='>', linecolor='red')
print(p)
```



16.5 Substitution

If you want to print ρ which takes more than one character, you can use an unused character in the string and then pass in a dictionary to replace the character to the actual string that you need.

```
from latextool_basic import *
p = Plot()

M = r"""
twice (fun x -> x * x) 42    #


(fun f -> (fun x -> f(f x))) (fun x -> x * x) 42
""".strip()

code(p, M, d={'#':r'\textred{$\rho$}'})
print(p)
```

```
twice (fun x -> x * x) 42    ρ

(fun f -> (fun x -> f(f x))) (fun x -> x * x) 42
```

16.6 Long division example

`linebelow` is used to draw lines in code. `divlinebelow` is used for long division line.

```
from latextool_basic import *
p = Plot()

M = r"""
.    01
1101 101010101010
    -0000
     10101
     -1101
      10000
       1101
""".strip()

N = code(p, M)
divlinebelow(p, N, 0, 5, 16)
linebelow(p, N, 2, 4, 8)
linebelow(p, N, 4, 5, 9)
print(p)
```

```
         01
    1101)101010101010
        -0000
         10101
         -1101
          10000
           1101
```

debug: table.

17 Tables

17.1 `table` function

```
from latextool_basic import table
print(table({0:0, 1:1, 2:4, 3:9, 4:16}))
```

0	0
1	1
2	4
3	9
4	16

Too few row headings:

```
from latextool_basic import table
print(table([('',  2, 5,  8),
             ('', '', 3,  6),
             ('', '', '', 3),
             ('', '', '', '')
            ],
            col_headings = ['0', '2', '5', '8'],
            row_headings = ['0', '2'],
            topleft_heading = r'$\Delta X$',
           ))
```

ΔX	0	2	5	8
0		2	5	8
2			3	6
				3

Too few col headings

```
from latextool_basic import table
print(table([('',  2, 5,  8),
             ('', '', 3,  6),
             ('', '', '', 3),
             ('', '', '', '')
            ],
            col_headings = ['0', '2'],
            row_headings = ['0', '2', '5', '8'],
            topleft_heading = r'$\Delta X$',
           ))
```

ΔX	0	2		
0		2	5	8
2			3	6
5				3
8				

Too few rows of data

```
from latextool_basic import table
print(table([('',  2, 5,  8),
             ('', '', 3,  6),
            ],
            col_headings = ['0', '2', '5', '8'],
            row_headings = ['0', '2', '5', '8'],
            topleft_heading = r'$\Delta X$',
           ))
```

ΔX	0	2	5	8
0		2	5	8
2			3	6
5				
8				

Too few rows of data

```
from latextool_basic import table
print(table([('',  2, 5,  8),
             ('', '', 3,  6),
            ],
            col_headings = ['0', '2', '5', '8'],
            row_headings = ['0', '2', '5', '8'],
            topleft_heading = r'$\Delta X$',
           ))
```

ΔX	0	2	5	8
0		2	5	8
2			3	6
5				
8				

Column headings only (no row headings). Note that topleft heading does not show.

```
from latextool_basic import table
print(table([('',  2, 5,),
             ('', '', 3,),
             ('', '', '',),
             ('', '', '',)
            ],
            col_headings = ['0', '2', '5', '8'],
            topleft_heading = r'$\Delta X$',
           ))
```

0	2	5	8
	2	5	
		3	

Row headings only (no column headings). Note that topleft heading does not show.

```
from latextool_basic import table
print(table([('',  2, 5,),
             ('', '', 3,),
             ('', '', '',),
             ('', '', '',)
            ],
            row_headings = ['0', '2', '5', '8'],
            topleft_heading = r'$\Delta X$',
           ))
```

0		2	5
2			3
5			
8			

Setting column justification.

```
from latextool_basic import table
print(table([('AAAAAAAAAA',  'BBBBBBBBB', 'CCCCCCCCC',),
             ('1', '2', 3,),
             ('', '', '',),
             ('', '', '',)
            ],
            row_headings = ['0', '2', '5', '8'],
            col_headings = ['a', 'b', 'c'],
            topleft_heading = r'$\Delta X$',
            style='|c|l|r|c|',
           ))
```

ΔX	a	b	c
0	AAAAAAAAAA	BBBBBBBBB	CCCCCCCCC
2	1	2	3
5			
8			

Right row headings

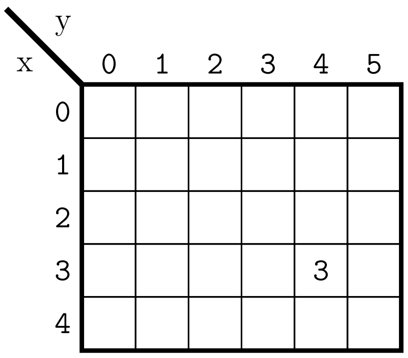
```
from latextool_basic import table
print(table([('',  2, 5,),
             ('', '', 3,),
             ('', '', '',),
             ('', '', '',)
            ],
            right_row_headings = ['0', '2', '5', '8'],
            col_headings = ['a', 'b', 'c'],
           ))
```

a	b	c
	2	5
		3

debug: table2

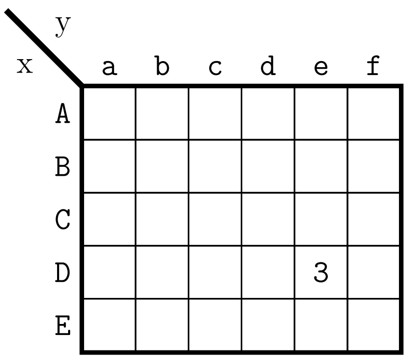
17.2 table2 function

```
from latextool_basic import *
p = Plot()

m = [['','','','','',''],
     ['','','','','',''],
     ['','','','','',''],
     ['','','','','3',''],
     ['','','','','',''],
     ]

table2(p, m, width=0.7, height=0.7, rowlabel='x', collabel='y')
print(p)
```



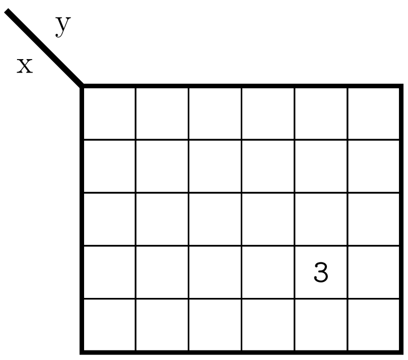
17.2.1 Change row and column index values/names

```
from latextool_basic import *
p = Plot()

m = [['','','','','',''],
     ['','','','','',''],
     ['','','','','',''],
     ['','','','','3',''],
     ['','','','','',''],
     ]

table2(p, m, width=0.7, height=0.7, rowlabel='x', collabel='y',
       rownames=[r'\texttt{%s}' % _ for _ in "ABCDE"],
       colnames=[r'\texttt{%s}' % _ for _ in "abcdef"])
print(p)
```



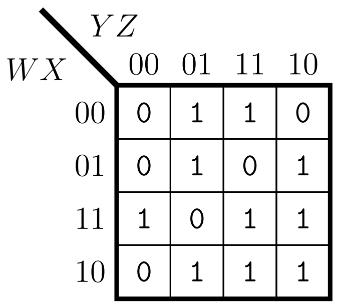
17.2.2 No row and column indices/values

```
from latextool_basic import *
p = Plot()

m = [['','','','','',''],
     ['','','','','',''],
     ['','','','','',''],
     ['','','','','3',''],
     ['','','','','',''],
     ]

table2(p, m, width=0.7, height=0.7, rowlabel='x', collabel='y',
       rownames=[],
       colnames=[])
print(p)
```



17.2.3 Change row/column names

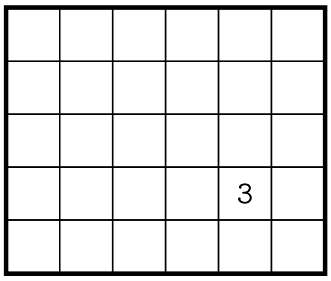
```
from latextool_basic import *

p = Plot()

m = [[0,1,1,0],
     [0,1,0,1],
     [1,0,1,1],
     [0,1,1,1],
     ]

table2(p, m, width=0.7, height=0.7,
       rownames=['00','01','11','10'],
       colnames=['00','01','11','10'],
       rowlabel='$WX$', collabel='$YZ$')
print(p)
```



WX \ YZ	00	01	11	10
00	0	1	1	0
01	0	1	0	1
11	1	0	1	1
10	0	1	1	1

17.2.4 No row and no column names

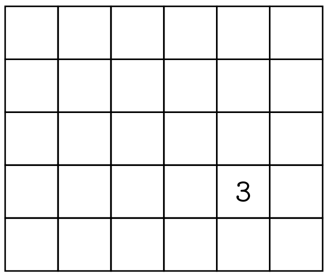
```
from latextool_basic import *
p = Plot()

m = [['','','','','',''],
     ['','','','','',''],
     ['','','','','',''],
     ['','','','','3',''],
     ['','','','','',''],
     ]

table2(p, m, width=0.7, height=0.7,
       rownames=[],
       colnames=[],
       rowlabel=None, collabel=None)
print(p)
```

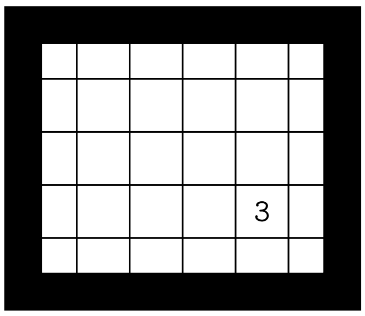


				3	

17.2.5 border_linewidth

```
from latextool_basic import *
p = Plot()

m = [['','','','','',''],
     ['','','','','',''],
     ['','','','','',''],
     ['','','','','3',''],
     ['','','','','',''],
     ]

table2(p, m, x=0, y=0, width=0.7, height=0.7,
       rownames=[],
       colnames=[],
       rowlabel=None, collabel=None, border_linewidth=0)
table2(p, m, x=8, y=0, width=0.7, height=0.7,
       rownames=[],
       colnames=[],
       rowlabel=None, collabel=None, border_linewidth=0.5)
print(p)
```

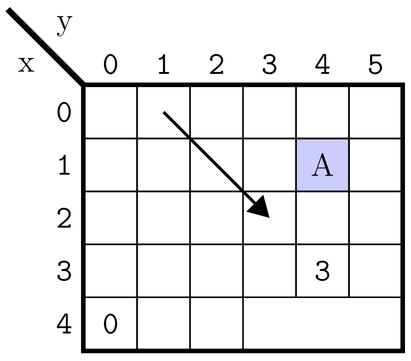




17.2.6 Adding things beneath table2: background

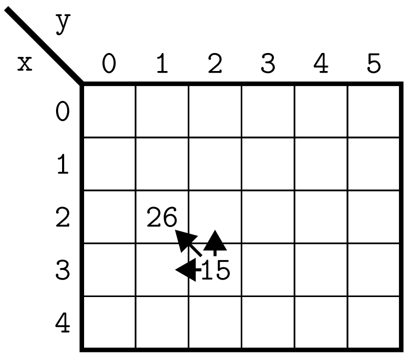
The `do_not_plot` can be used to compute the coordinate of the cells. `table2` will then return the RectContainer object but the table is not drawn. So one way to change the background is to do `do_not_plot` to get the coordinates, draw a rect with background color, then call `table2` to draw the contents of the table. (Also, see rect parameter later.)

```
from latextool_basic import *

m = [['','','','','',''],
     ['','','','','',''],
     ['','','','','',''],
     ['','','','','3',''],
     ['0','',''],           # OK to have jagged arrays
     ]

p = Plot()
C = table2(p, m, rowlabel='x', collabel='y', do_not_plot=True)

# Draw over C[1][4]
x0,y0 = C[1][4].bottomleft()
x1,y1 = C[1][4].topright()
p += Rect(x0=x0, y0=y0, x1=x1, y1=y1,
          background='blue!20',label='A', linewidth=0)

# Draw table one more time to get border of C[1][4] correct
table2(p, m, rowlabel='x', collabel='y')

p += Line(points=[C[0][1].center(), C[2][3].center()], endstyle='>')

print(p)
```



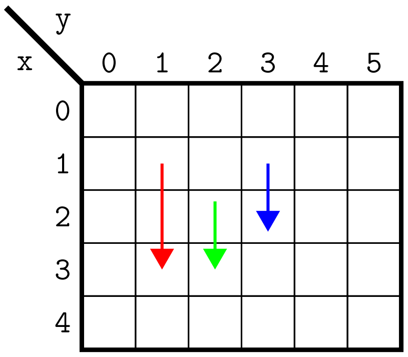
17.2.7 Using cells

Drawing arrows between cells using `shorten` function for arrows using a shortening factor:



```
x = "     "
y = "      "
m = [['' for i in range(len(y))] for j in range(len(x))]
m[3][2] = 15
m[2][1] = 26
from latextool_basic import *
p = Plot()
C = table2(p, m, width=0.7, height=0.7,
           rowlabel=r'\texttt{x}', collabel=r'\texttt{y}')

p0 = C[3][2].center()
p1 = C[2][1].center()
p0, p1 = shorten(p0, p1, factor=0.5)
p += Line(points=[p0, p1], endstyle='>')

p0 = C[3][2].center()
p1 = C[2][2].center()
p0, p1 = shorten(p0, p1, factor=0.5)
p += Line(points=[p0, p1], endstyle='>')

p0 = C[3][2].center()
p1 = C[3][1].center()
p0, p1 = shorten(p0, p1, factor=0.5)
p += Line(points=[p0, p1], endstyle='>')

print(p)
```

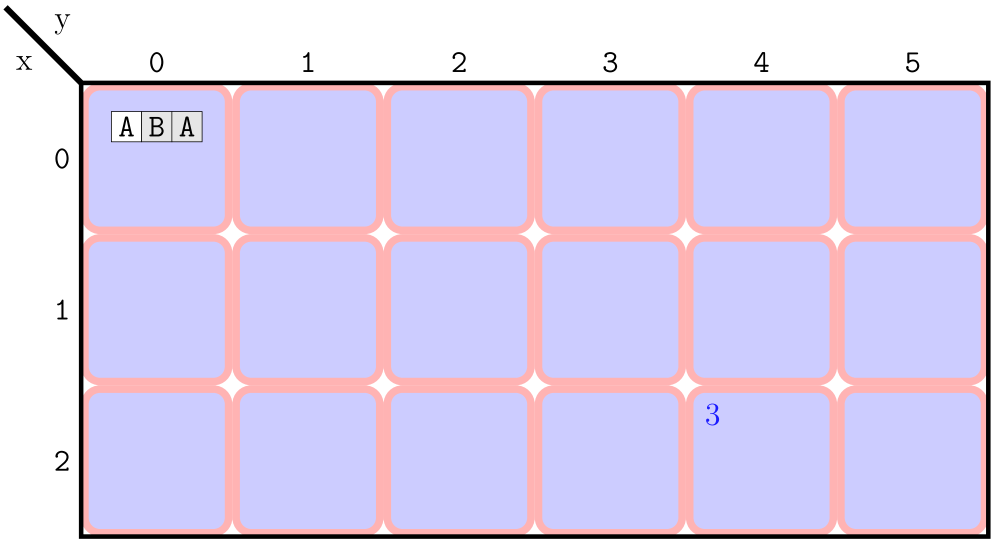
Drawing arrows between cells using `shorten` function for arrows using a shortening amount:



```
x = "     "
y = "      "
m = [['' for i in range(len(y))] for j in range(len(x))]
from latextool_basic import *
p = Plot()
C = table2(p, m, width=0.7, height=0.7,
           rowlabel=r'\texttt{x}', collabel=r'\texttt{y}')

p0 = C[1][1].center()
p1 = C[3][1].center()
p += Line(points=[p0, p1], endstyle='>', linecolor='red')

p0 = C[1][2].center()
p1 = C[3][2].center()
p0, p1 = shorten(p0, p1, start_by=0.5)
p += Line(points=[p0, p1], endstyle='>', linecolor='green')

p0 = C[1][3].center()
p1 = C[3][3].center()
p0, p1 = shorten(p0, p1, end_by=0.5)
p += Line(points=[p0, p1], endstyle='>', linecolor='blue')

print(p)
```

17.2.8 Rect for cells

You can specify a rect function of your own:

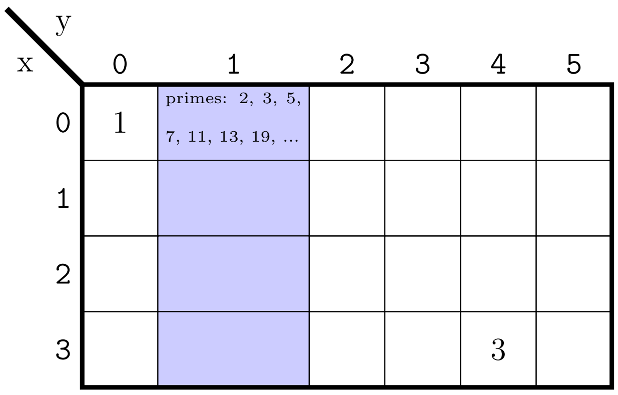


```
from latextool_basic import *

c = RectContainer(x=0, y=0)
xs = 'ABA'; shadedbackground='black!10'
for i,x in enumerate(xs):
    if i in [1,2]:
        c += Rect2(x0=0, y0=0, x1=0.4, y1=0.4,
             background=shadedbackground,
             linewidth=0.01, label=r'{\texttt{%s}}' % x)
    else:
        c += Rect2(x0=0, y0=0, x1=0.4, y1=0.4,
                   background='white',
                   linewidth=0.01, label=r'{\texttt{%s}}' % x)
q = Plot(); q += c
m = [[str(q),'','','','',''],
     ['','','','','',''],
     ['','','','','3',''],
     ]

p = Plot()

def rect(x):
    return Rect(x0=0, y0=0, x1=2, y1=2, radius=0.2, innersep=0.2,
    linecolor='red!30!white', linewidth=0.1,
    background='blue!20!white!', foreground='blue!90!white',
    s='%s' % x, align='t')

table2(p, m, width=0.7, height=0.7, rowlabel='x', collabel='y', rect=rect)
print(p)
```

17.2.9 Using rect to change the background

You can use the rect to for instance change the background. In the following, the rects at column 2 are given larger width and a different background, and the contents are placed in a minipage with tiny fontsize.

```
from latextool_basic import *
m = [['1','primes: 2, 3, 5, 7, 11, 13, 19, ...','','','',''],
     ['','','','','',''],
     ['','','','','',''],
     ['','','','','3',''],
     ]
def get_rect():
    i = [0]
    def rect(x):
        if i[0] % 6 == 1:
            i[0] += 1
            return Rect(x0=0, y0=0, x1=2, y1=1,
                        background='blue!20!white!',
                        innersep=0.1,
                        s=r'{\tiny %s}' % x, align='t')
        else:
            i[0] += 1
            return Rect(x0=0, y0=0, x1=1, y1=1, label=x)
    return rect
p = Plot()
table2(p, m, rowlabel='x', collabel='y', rect=get_rect())
print(p)
```



debug: table3

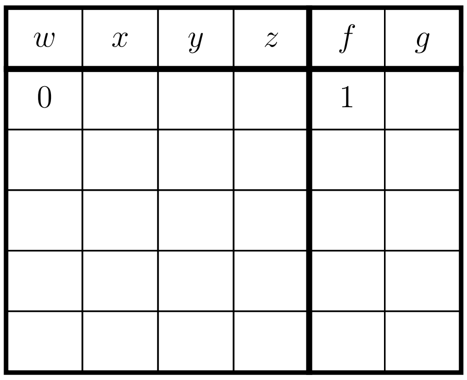
17.3 table3

table3 ... TABLE OF 2d arrays of 2d arrays:

Use this for truth tables.

17.3.1 Example: 2-by-2 case

```
from latextool_basic import *
p = Plot()

m00 = [['$w$','$x$','$y$','$z$']]
m10 = [['$0$','','',''],
       ['','','',''],
       ['','','',''],
       ['','','',''],
       ['','','','']]
m01 = [['$f$','$g$'],
      ]
m11 =[['$1$',''],
      ['',''],
      ['',''],
      ['',''],
      ['','']]
M = [[m00, m01],
     [m10, m11]]
N = table3(p, M, width=1, height=0.8)
print(p)
```



w	x	y	z	f	g
0				1	

17.3.2 Example: 2-by-2 case with top and left for labels

```
from latextool_basic import *
p = Plot()

rowlabels = ['A', 'B', 'C', 'D', 'E']
collabels = ['a', 'b', 'c']
data = [['00', '01', '02'],
        ['10', '11', '12'],
        ['20', '21', '22'],
        ['30', '31', '32'],
        ['40', '41', '42']]

m00 = [['']]
m10 = [[x] for x in rowlabels]
m01 = [collabels]
m11 = data
M = [[m00, m01],
     [m10, m11]]
N = table3(p, M, width=1, height=0.8)
print(p)
```

	a	b	c
A	00	01	02
B	10	11	12
C	20	21	22
D	30	31	32
E	40	41	42

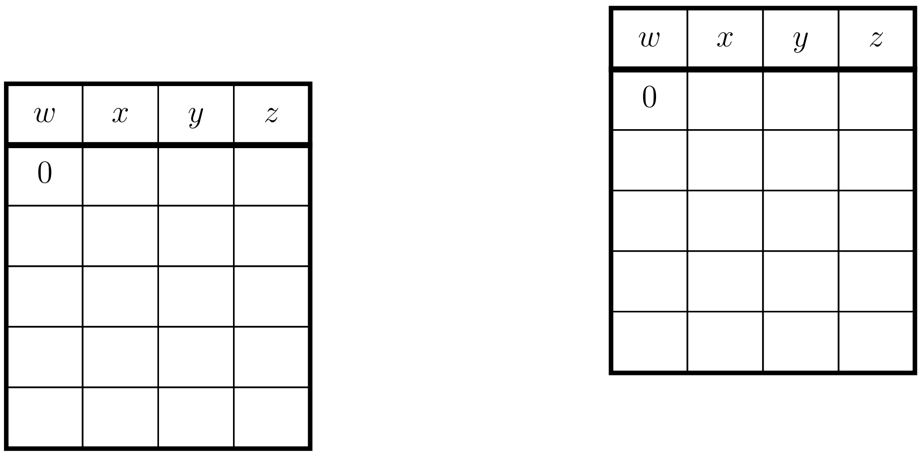
17.3.3 Example: 2-by-1 case

```
from latextool_basic import *
p = Plot()

m00 = [['$w$','$x$','$y$','$z$'],
      ]
m10 = [['$0$','','',''],
       ['','','',''],
       ['','','',''],
       ['','','',''],
       ['','','',''],
      ]
M = [[m00],
     [m10]]
N = table3(p, M, width=1, height=0.8)
print(p)
```

w	x	y	z
0			

17.3.4 Specify topleft coordinates

```
from latextool_basic import *
p = Plot()

m00 = [['$w$','$x$','$y$','$z$'],
      ]
m10 = [['$0$','','',''],
       ['','','',''],
       ['','','',''],
       ['','','',''],
       ['','','',''],
      ]
M = [[m00],
     [m10]]
table3(p, x0=0, y0=0, M=M, width=1, height=0.8)
table3(p, x0=8, y0=1, M=M, width=1, height=0.8)
print(p)
```

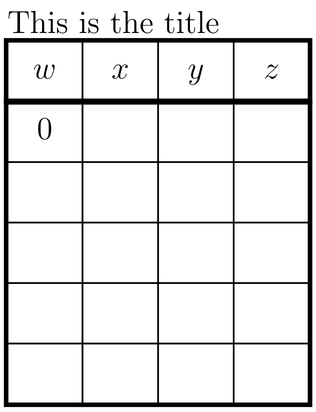


w	x	y	z
0			

w	x	y	z
0			

17.3.5 Example: 2-by-1 case with title

```
from latextool_basic import *
p = Plot()

m00 = [['$w$','$x$','$y$','$z$'],
      ]
m10 = [['$0$','','',''],
       ['','','',''],
       ['','','',''],
       ['','','',''],
       ['','','',''],
      ]
M = [[m00],
     [m10]]
table3(p, M, width=1, height=0.8, title='This is the title')
print(p)
```

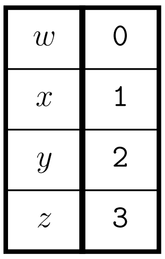


This is the title

w	x	y	z
0			

17.3.6 Example: 1-by-2 case

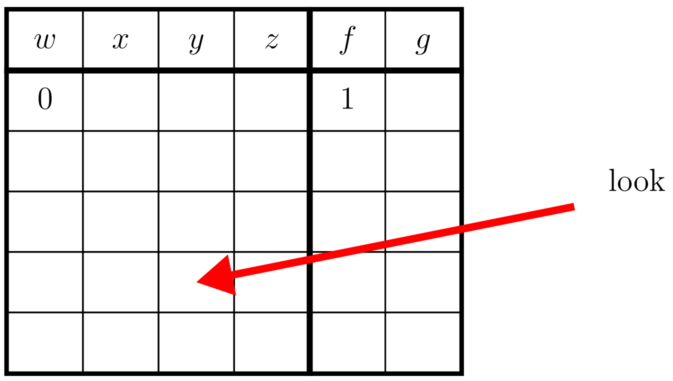
```
from latextool_basic import *
p = Plot()

m00 = [['$w$'],['$x$'],['$y$'],['$z$'],
      ]
m01 = [['0'],['1'],['2'],['3'],
      ]
M = [[m00, m01],
     ]
N = table3(p, M, width=1, height=0.8)
print(p)
```



w	0
x	1
y	2
z	3

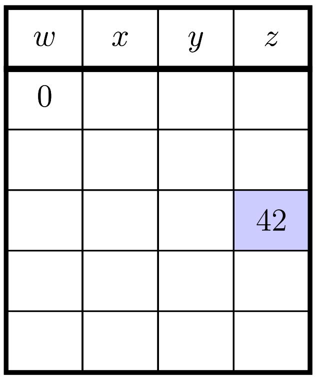
17.3.7 table3: accessing cells

```
from latextool_basic import *
p = Plot()

m00 = [['$w$','$x$','$y$','$z$'],
      ]
m10 = [['$0$','','',''],
       ['','','',''],
       ['','','',''],
       ['','','',''],
       ['','','',''],
      ]
m01 = [['$f$','$g$'],
      ]
m11 =[['$1$',''],
      ['',''],
      ['',''],
      ['',''],
      ['',''],
      ]
M = [[m00, m01],
     [m10, m11]]
N = table3(p, M, width=1, height=0.8)
p0 = x0,y0 = N[1][0][3][2].center()
p1 = x1,y1 = x0 + 5, y0 + 1
p += Line(points=[p1, p0], endstyle='>', linecolor='red', linewidth=0.1)
x2,y2 = x1+0.5, y1+0.5
p += Rect(x0=x2, y0=y0, x1=x2, y1=y2, s='look', linewidth=0)
print(p)
```



17.3.8 Coloring a cell under the table manually

```
from latextool_basic import *
p = Plot()

m00 = [['$w$','$x$','$y$','$z$'],
      ]
m10 = [['$0$','','',''],
       ['','','',''],
       ['','','',''],
       ['','','',''],
       ['','','',''],
      ]
M = [[m00],
     [m10]]

N = table3(p, M, width=1, height=0.8)

r = N[1][0][2][3]
x0,y0 = r.bottomleft()
x1,y1 = r.topright()
p += Rect(x0=x0, y0=y0, x1=x1, y1=y1, background='blue!20', label='42',
          linewidth=0)

N = table3(p, M, width=1, height=0.8)

print(p)
```



17.3.9 Coloring a cell under the table using table3

```
from latextool_basic import *
p = Plot()

m00 = [['$w$','$x$','$y$','$z$'],
      ]
m10 = [['$0$','','',''],
       ['','','',''],
       ['','','','42'],
       ['','','',''],
       ['','','',''],
      ]
M = [[m00],
     [m10]]

N = table3(p, M, width=2, height=0.5,
           background={(1,0,2,3):'blue!20'})

print(p)
```

w	x	y	z
0			
			42

17.3.10 Cells of different widths (table2)

```
from latextool_basic import *
p = Plot()

i = 0
def rect(x):
    global i
    if i % 2 == 0:
        i += 1
        return Rect(x0=0, y0=0, x1=1, y1=0.7, s=x)
    else:
        i += 1
        return Rect(x0=0, y0=0, x1=3, y1=0.7, s=x)

m = [[0,1,2,3],
     [4,5,6,7],
     [8,9,10,11],
     [12,13,14,15],
     ]

table2(p, m, rect=rect)
print(p)
```

17.3.11 Cells of different widths (table3)

```
from latextool_basic import *
p = Plot()

i = 0
def rect(x):
    global i
    if i in [0,3,6,8,10,13,16,19,21,23]:
        i += 1; return Rect(x0=0, y0=0, x1=1, y1=0.7, label=x)
    elif i in [1,4,7,9,11,14,17,20,22,24]:
        i += 1; return Rect(x0=0, y0=0, x1=3, y1=0.7, label=x)
    else:
        i += 1; return Rect(x0=0, y0=0, x1=5, y1=0.7, label=x)

m00 = [[0,1,2],
       [3,4,5]]
m01 = [[6,7],
       [8,9]]
m10 = [[10,11,12],
       [13,14,15],
       [16,17,18]]
m11 = [[19,20],
       [21,22],
       [23,24]]
M = [[m00, m01],
     [m10, m11]]
table3(p, M, rect=rect)
print(p)
```

0	1	2	6	7
3	4	5	8	9
10	11	12	19	20
13	14	15	21	22
16	17	18	23	24

17.3.12 ragged arrays (error - to fix)

```
from latextool_basic import *
p = Plot()

i = 0
def rect(x):
    global i
    if i in [0,3,6,8,10,13,16,19,21,23]:
        i += 1; return Rect(x0=0, y0=0, x1=1, y1=0.7, label=x)
    elif i in [1,4,7,9,11,14,17,20,22,24]:
        i += 1; return Rect(x0=0, y0=0, x1=3, y1=0.7, label=x)
    else:
        i += 1; return Rect(x0=0, y0=0, x1=5, y1=0.7, label=x)

m00 = [[0,1,2],
       [3,4,5]]
m01 = [[6,7],
       [8,9]]
m10 = [[10,11,12],
       [13,14,15],
       [16,17,18]]
m11 = [[19,20],
       [21,22],
       [23,24]]
M = [[m00, m01],
     [m10, m11]]
table3(p, M, rect=rect)
print(p)
```

0	1	2	6	7
3	4	5	8	9
10	11	12	19	20
13	14	15	21	22
16	17	18	23	24

17.3.13 Justification

```
from latextool_basic import *
p = Plot()

i = 0
def rect(x):
    global i
    if i in [0,3,6,8,10,13,16,19,21,23]:
        i += 1; return Rect(x0=0, y0=0, x1=1, y1=0.7, label=x)
    elif i in [1,4,7,9,11,14,17,20,22,24]:
        i += 1; return Rect(x0=0, y0=0, x1=3, y1=0.7, label=x)
    else:
        i += 1; return Rect(x0=0, y0=0, x1=5, y1=0.7, label=x)

m00 = [[0,1,2],
       [3,4,5]]
m01 = [[6,7],
       [8,9]]
m10 = [[10,11,12],
       [13,14,15],
       [16,17,18]]
m11 = [[19,20],
       [21,22],
       [23,24]]
M = [[m00, m01],
     [m10, m11]]
table3(p, M, rect=rect)
print(p)
```

0	1	2	6	7
3	4	5	8	9
10	11	12	19	20
13	14	15	21	22
16	17	18	23	24

17.3.14 title

```
from latextool_basic import *
p = Plot()

i = 0
def rect(x):
    global i
    i += 1
    return Rect(x0=0, y0=0, x1=1.8, y1=0.8, label=x)

m00 = [['A','B','C']]

s = r"""
1 2 3
4 5 6
7 8 9
"""
X = str_to_2darray(s)
M = [[[X[0]]],
     [X[1:]],
     ]
table3(p, M, rect=rect, title='This is a title')
print(p)
```

This is a title

1	2	3
4	5	6
7	8	9

17.3.15 Title distance

```
from latextool_basic import *
p = Plot()

i = 0
def rect(x):
    global i
    i += 1
    return Rect(x0=0, y0=0, x1=1.8, y1=0.8, label=x)

m00 = [['A','B','C']]

s = r"""
1 2 3
4 5 6
7 8 9
"""
X = str_to_2darray(s)
M = [[[X[0]]],
     [X[1:]],
    ]
table3(p, M, rect=rect, title='This is a title', title_distance=0.8)
print(p)
```

This is a title

1	2	3
4	5	6
7	8	9

debug: table4

17.4 table4

```
from latextool_basic import *
p = Plot()
M = [['a', 'b', 'c'],
     [0, 1, 2],
     [3, 4, 5],
     [6, 7, 8]]
widths=[0.8, 1.5, 3]
N = table4(p=p, M=M, widths=widths)
print(p)
```

a	b	c
0	1	2
3	4	5
6	7	8

title distance:

```
from latextool_basic import *
p = Plot()
M = [['a', 'b', 'c'],
     [0, 1, 2],
     [3, 4, 5],
     [6, 7, 8]]
widths=[0.8, 1.5, 3]
N = table4(p=p, M=M, widths=widths, title='This is the title')
print(p)
```

This is the title

a	b	c
0	1	2
3	4	5
6	7	8

make caption:

```
from latextool_basic import *
from latexcircuit import *
p = Plot()
M = [['a', 'b', 'c'],
     [0, 1, 2],
     [3, 4, 5],
     [6, 7, 8]]
widths=[0.8, 1.5, 3]
N = table4(p=p, M=M, widths=widths, title='This is the title')
x0,y0 = N[-1][0].bottomleft()
x1,y1 = N[-1][-1].bottomright()
y = y0
x = (x0 + x1) / 2
p += POINT(x=x, y=y, r=0, label='This is the caption', anchor='north')
print(p)
```

This is the title

a	b	c
0	1	2
3	4	5
6	7	8

17.5 helper for 2d arrays

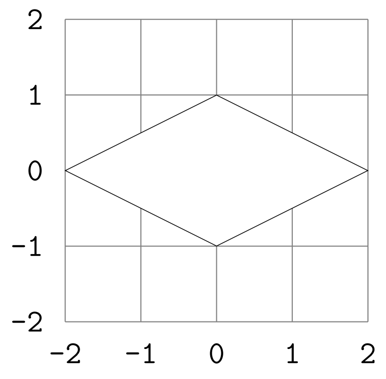
```
from latextool_basic import *
p = Plot()
M = r"""
a b c
0 1 2
3 4 5
6 7 8
"""
M = str_to_2darray(M)
widths=[0.8, 1.5, 3]
N = table4(p=p, M=M, widths=widths)
print(p)
```

a	b	c
0	1	2
3	4	5
6	7	8

debug: diamc

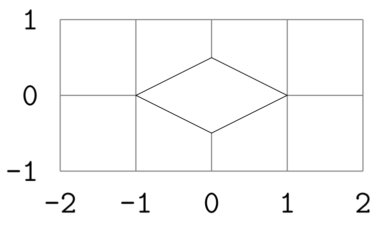
18 diamond

```
from latextool_basic import *
p = Plot()
p += Grid(-2,-2,2,2)
p += diamond(center=(0,0))
print(p)
```

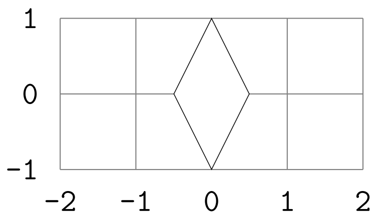


Width, height:

```
from latextool_basic import *
p = Plot()
p += Grid(-2,-1,2,1)
p += diamond(center=(0,0), width=2, height=1)
print(p)
```

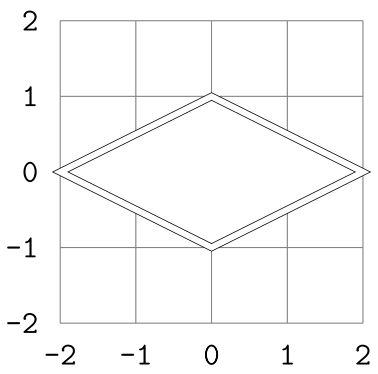


```
from latextool_basic import *
p = Plot()
p += Grid(-2,-1,2,1)
p += diamond(center=(0,0), width=1, height=2)
print(p)
```



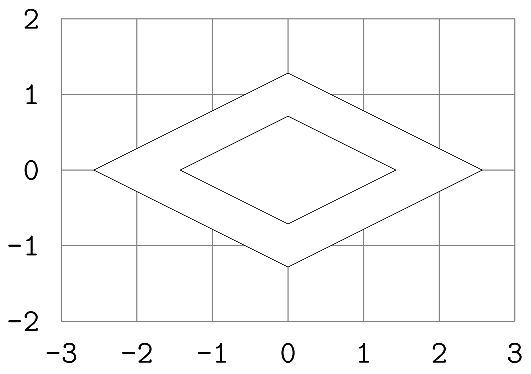
double line:

```
from latextool_basic import *
p = Plot()

p += Grid(-2,-2,2,2)
p += diamond(center=(0,0), double=True)
print(p)
```



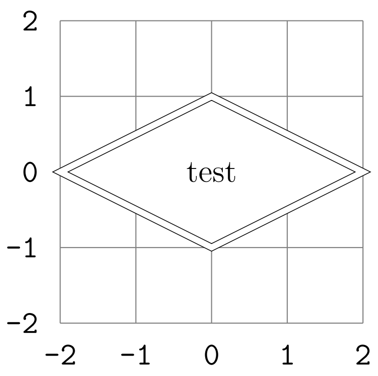
double line with double distance

```
from latextool_basic import *
p = Plot()

p += Grid(-3,-2,3,2)
p += diamond(center=(0,0), double=True, double_distance=0.5)
print(p)
```

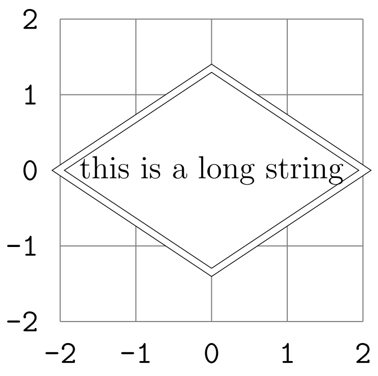


double and label

```
from latextool_basic import *
p = Plot()

p += Grid(-2,-2,2,2)
p += diamond(center=(0,0), double=True, label='test')
print(p)
```



```
from latextool_basic import *
p = Plot()

p += Grid(-2,-2,2,2)
p += diamond(center=(0,0), double=True, label='this is a long string')
print(p)
```

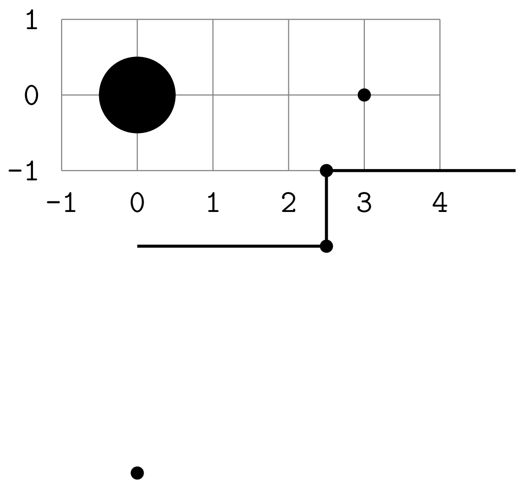


debug: circle.

19 Circle

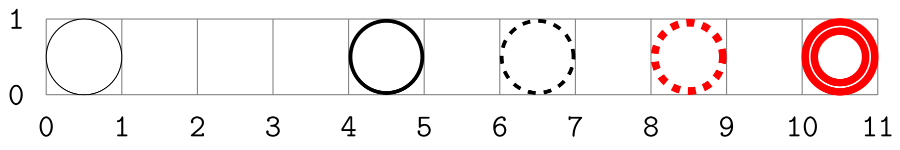
19.1 boundary

```
from latextool_basic import *
from latexcircuit import *

p = Plot()
p += Circle(x=0, y=0, r=0.5, background='black')
p += Circle(x=3, y=0, r=0.08, background='black', name=None)


p += str(OrthogonalPath([(0,-2), (5,-1)]))

p += POINT(x=0, y=-3)
p += '%s' % Circle(x=0, y=-5, r=0.08, background='black', name=None)

p += Grid()
print(p)
```



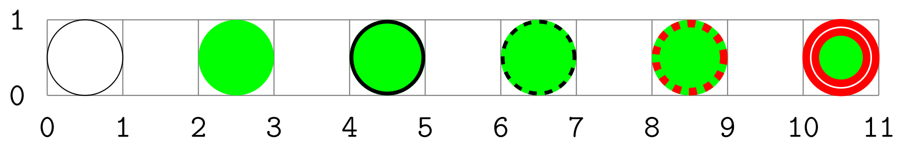
19.2 boundary

```
from latextool_basic import *
p = Plot()
p += Circle(x=0.5, y=0.5, r=0.5)
p += Circle(x=2.5, y=0.5, r=0.5, linewidth=0)
p += Circle(x=4.5, y=0.5, r=0.5, linewidth=0.05)
p += Circle(x=6.5, y=0.5, r=0.5, linewidth=0.05, linestyle='dashed')
p += Circle(x=8.5, y=0.5, r=0.5, linewidth=0.1, linestyle='dashed',
                                 linecolor='red')
p += Circle(x=10.5, y=0.5, r=0.5, linewidth=0.1, linestyle='double',
                                 linecolor='red')
p += Grid()
print(p)
```



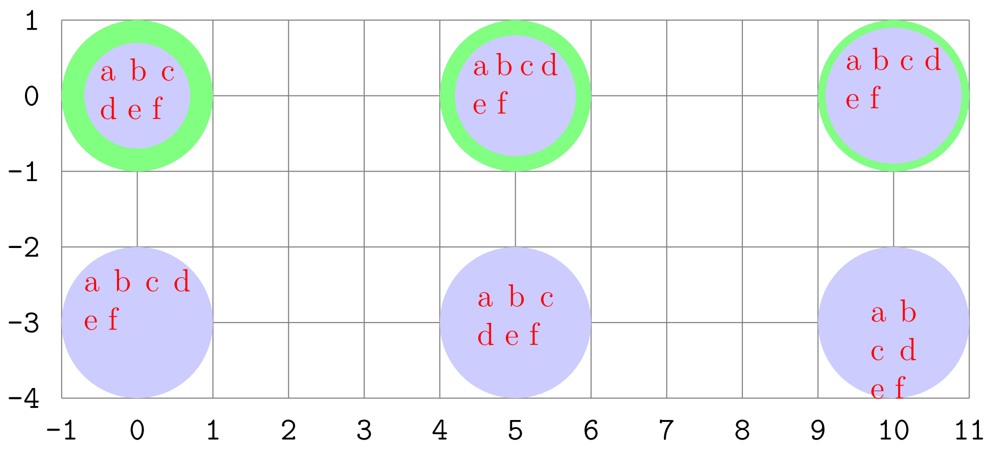
19.3 background

```
from latextool_basic import *
p = Plot()
p += Circle(x=0.5, y=0.5, r=0.5)
p += Circle(x=2.5, y=0.5, r=0.5, linewidth=0, background='green')
p += Circle(x=4.5, y=0.5, r=0.5, linewidth=0.05, background='green')
p += Circle(x=6.5, y=0.5, r=0.5, linewidth=0.05, linestyle='dashed',
                                 background='green')
p += Circle(x=8.5, y=0.5, r=0.5, linewidth=0.1, linestyle='dashed',
                                 linecolor='red', background='green')
p += Circle(x=10.5, y=0.5, r=0.5, linewidth=0.1, linestyle='double',
                                 linecolor='red', background='green')
p += Grid()
print(p)
```



19.4 s: minipage text in circle

```
from latextool_basic import *
p = Plot()
s = 'a b c d e f'
p += Circle(x=0, y=0, r=1, linecolor='green!50!white', linewidth=0.3,
            background='blue!20!white', foreground='red',innersep=0, s=s)
p += Circle(x=5, y=0, r=1, linecolor='green!50!white', linewidth=0.2,
            background='blue!20!white', foreground='red', innersep=0, s=s)
p += Circle(x=10, y=0, r=1, linecolor='green!50!white', linewidth=0.1,
            background='blue!20!white', foreground='red', innersep=0, s=s)
p += Circle(x=0, y=-3, r=1, linecolor='green!50!white', linewidth=0,
            background='blue!20!white', foreground='red', innersep=0, s=s)
p += Circle(x=5, y=-3, r=1, linecolor='green!50!white', linewidth=0,
            background='blue!20!white', foreground='red',
            innersep=0.2, s=s)
p += Circle(x=10, y=-3, r=1, linecolor='green!50!white', linewidth=0,
            background='blue!20!white', foreground='red',
            innersep=0.4, s=s)
p += Grid()
print(p)
```



19.5 label anchor

```
from latextool_basic import *
p = Plot()
label = 'a'
p += Circle(x=0, y=0, r=1, label=label, anchor='above')
p += Grid()
print(p)
```

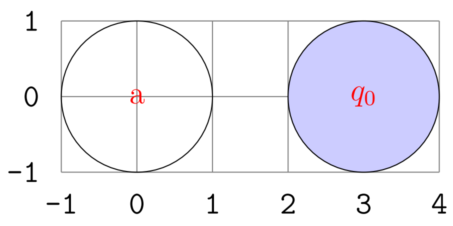
PYTHON ERROR. See source, stderr, stdout below

```
from latextool_basic import *
p = Plot()
label = 'a'
p += Circle(x=0, y=0, r=1, label=label, anchor='above')
p += Grid()
print(p)
```

```
Traceback (most recent call last):
  File "/usr/lib64/python3.7/site-packages/latextool_basic.py", line 715, i
n __str__
    shape, arg, karg = value
TypeError: cannot unpack non-iterable Circle object

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "/usr/lib64/python3.7/site-packages/latextool_basic.py", line 722, i
n __str__
    s += str(value)
  File "/usr/lib64/python3.7/site-packages/latextool_basic.py", line 1215,
in __str__
    name=name,
  File "/usr/lib64/python3.7/site-packages/latextool_basic.py", line 1154,
in circle
    % d1
KeyError: 'anchor'

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "ixilbqpe.tmp.py", line 6, in <module>
    print(p)
  File "/usr/lib64/python3.7/site-packages/latextool_basic.py", line 724, i
n __str__
    raise ValueError("str method in Plot error: %s, type:%s" % (value, type
(value)))
  File "/usr/lib64/python3.7/site-packages/latextool_basic.py", line 1215,
in __str__
    name=name,
  File "/usr/lib64/python3.7/site-packages/latextool_basic.py", line 1154,
in circle
    % d1
KeyError: 'anchor'
```

19.6 foreground

```
from latextool_basic import *
p = Plot()
p += Circle(x=0, y=0, r=1, foreground='red', label='a')
p += Circle(x=3, y=0, r=1, background='blue!20!white',
                           foreground='red', label='$q_0$')
p += Grid()
print(p)
```

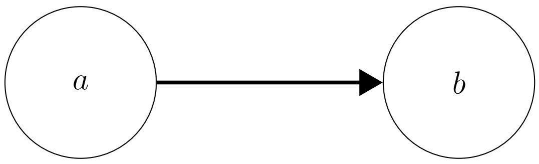


19.7 Center and boundary

```
s='1 2 3 4 5 6 7 8 9 10 11'
b1 = 'blue!50!white'
b2 = 'red!50!white'

from latextool_basic import *
p = Plot()

c1 = Circle(x=0, y=0, r=1, background=b1, s=s)
c2 = Circle(center=c1.center(), r=0.5, background=b2)
p += c1; p += c2

c1 = Circle(x=3, y=0, r=1, background=b1, s=s)
c2 = Circle(center=c1.left(), r=0.5, background=b2)
p += c1; p += c2

c1 = Circle(x=6, y=0, r=1, background=b1, s=s)
c2 = Circle(center=c1.right(), r=0.5, background=b2)
p += c1; p += c2

c1 = Circle(x=9, y=0, r=1, background=b1, s=s)
c2 = Circle(center=c1.top(), r=0.5, background=b2)
p += c1; p += c2

c1 = Circle(x=12, y=0, r=1, background=b1, s=s)
c2 = Circle(center=c1.bottom(), r=0.5, background=b2)
p += c1; p += c2

p += Grid()
print(p)
```

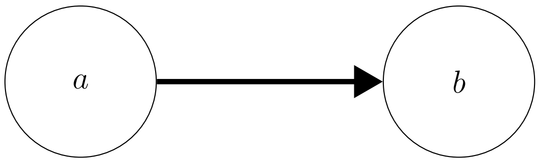
19.8 tikz name

```
from latextool_basic import *
p = Plot()

p += Circle(center=(0,0), r = 1, label=r'$a$', name='a')
p += Circle(center=(5,0), r = 1, label=r'$b$', name='b')

p += Line(names=['a', 'b'], linewidth=0.05, endstyle='>')

print(p)
```



Test with and without name:

```
from latextool_basic import *
p = Plot()

p += Circle(center=(0,0), r = 1, label=r'$a$', name='a')
p += Circle(center=(5,0), r = 1, label=r'$b$')

p += r"\draw[->,line width=2] (a) -- (b);"

print(p)
```

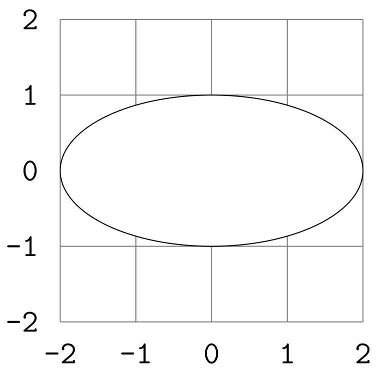


debug: ellipse

20 Ellipse

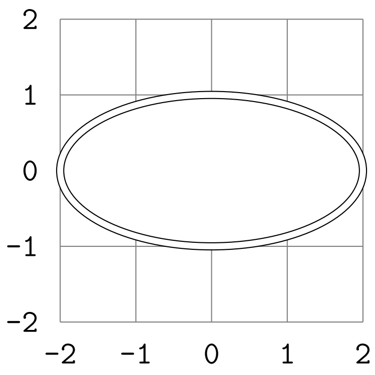
```
from latextool_basic import *
p = Plot()

p += Grid(-2,-2,2,2)
p += ellipse(-2,-1,2,1)
print(p)
```

```
from latextool_basic import *
p = Plot()

p += Grid(-2,-2,2,2)
p += ellipse(-2,-1,2,1)
print(p)
```



```
from latextool_basic import *
p = Plot()

p += Grid(-2,-2,2,2)
p += ellipse(-2,-1,2,1,double=True)
print(p)
```

```
from latextool_basic import *
p = Plot()

p += Grid(-2,-2,2,2)
p += ellipse(-2,-1,2,1,double=True)
print(p)
```

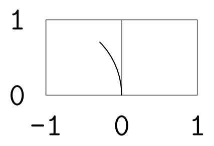


debug: arc.te

21 arc

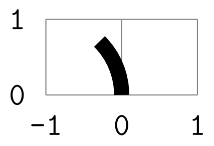
Draw part of a circle. The x, y is the starting point, NOT the center.

```
from latextool_basic import Plot, Grid, arc
p = Plot()
p += arc(x=0, y=0, r=1, angle0=0, angle1=45)
p += Grid(x0=-1, y0=0, x1=1, y1=1)
print(p)
```



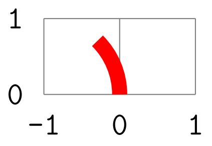
Linewidth:

```
from latextool_basic import Plot, Grid, arc
p = Plot()
p += arc(x=0, y=0, r=1, angle0=0, angle1=45, linewidth=0.2)
p += Grid(x0=-1, y0=0, x1=1, y1=1)
print(p)
```



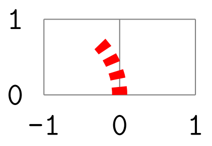
Color:

```
from latextool_basic import Plot, Grid, arc
p = Plot()
p += arc(x=0, y=0, r=1, angle0=0, angle1=45,
         linewidth=0.2, linecolor='red')
p += Grid(x0=-1, y0=0, x1=1, y1=1)
print(p)
```



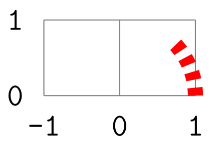
Linestyle:

```
from latextool_basic import Plot, Grid, arc
p = Plot()
p += arc(x=0, y=0, r=1, angle0=0, angle1=45,
        linewidth=0.2, linecolor='red', linestyle='dashed')
p += Grid(x0=-1, y0=0, x1=1, y1=1)
print(p)
```



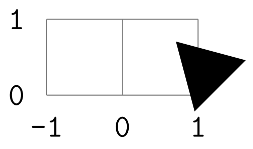
Change starting point of arc:

```
from latextool_basic import Plot, Grid, arc
p = Plot()
p += arc(x=1, y=0, r=1, angle0=0, angle1=45,
        linewidth=0.2, linecolor='red', linestyle='dashed')
p += Grid(x0=-1, y0=0, x1=1, y1=1)
print(p)
```

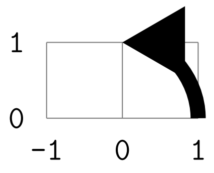


Arrow:

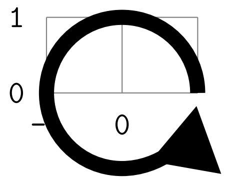
```
from latextool_basic import Plot, Grid, arc
p = Plot()
p += arc(x=1, y=0, r=1, angle0=0, angle1=45,
        linewidth=0.2, endstyle='>')
p += Grid(x0=-1, y0=0, x1=1, y1=1)
print(p)
```



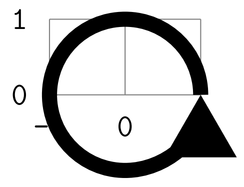
```
from latextool_basic import Plot, Grid, arc
p = Plot()
p += arc(x=1, y=0, r=1, angle0=0, angle1=90,
        linewidth=0.2, endstyle='>')
p += Grid(x0=-1, y0=0, x1=1, y1=1)
print(p)
```



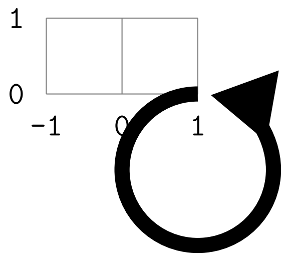
```
from latextool_basic import Plot, Grid, arc
p = Plot()
p += arc(x=1, y=0, r=1, angle0=0, angle1=350,
        linewidth=0.2, endstyle='>')
p += Grid(x0=-1, y0=0, x1=1, y1=1)
print(p)
```



```
from latextool_basic import Plot, Grid, arc
p = Plot()
p += arc(x=1, y=0, r=1, angle0=0, angle1=360,
        linewidth=0.2, endstyle='>')
p += Grid(x0=-1, y0=0, x1=1, y1=1)
print(p)
```

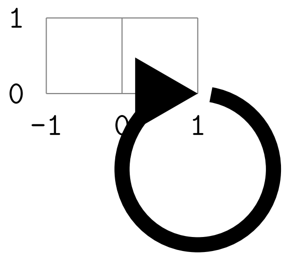


```
from latextool_basic import Plot, Grid, arc
p = Plot()
p += arc(x=1, y=0, r=1, angle0=90, angle1=360+80,
        linewidth=0.2, endstyle='>')
p += Grid(x0=-1, y0=0, x1=1, y1=1)
print(p)
```



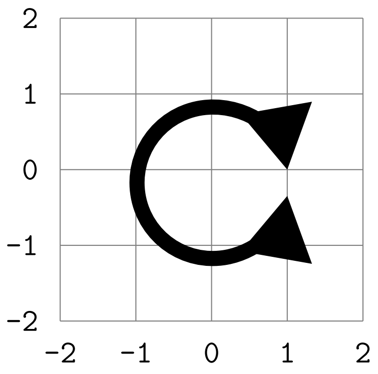
Start style

```
from latextool_basic import Plot, Grid, arc
p = Plot()
p += arc(x=1, y=0, r=1, angle0=90, angle1=360+80,
        linewidth=0.2, startstyle='>')
p += Grid(x0=-1, y0=0, x1=1, y1=1)
print(p)
```



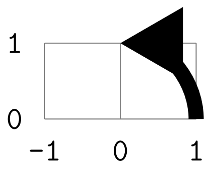
Start and end style

```
from latextool_basic import Plot, Grid, arc
p = Plot()
p += arc(x=1, y=0, r=1, angle0=10, angle1=350,
        linewidth=0.2, startstyle='>', endstyle='>')
p += Grid(x0=-2, y0=-2, x1=2, y1=2)
print(p)
```



With center:

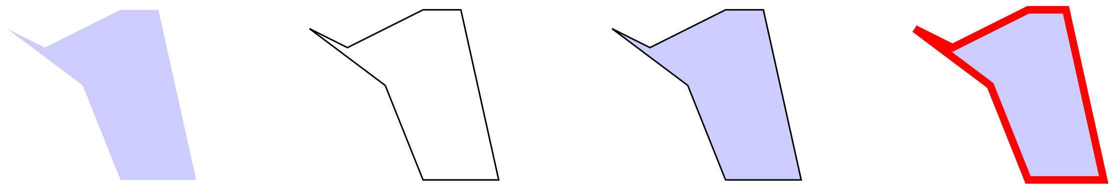
```
from latextool_basic import Plot, Grid, arc
p = Plot()
p += arc(center=(0,0), r=1, angle0=0, angle1=90,
        linewidth=0.2, endstyle='>')
p += Grid(x0=-1, y0=0, x1=1, y1=1)
print(p)
```



debug: polyg

22 Polygon

```
import random; random.seed()
from latextool_basic import *

def IFELSE(b, x, y):
    if b: return x
    else: return y

def polygon(points,
            background=None, # if background if None, no fill
            linewidth=None,
            linecolor=None,
            ):
    if points[-1] != points[0]: points.append(points[0])
    points_str = " -- ".join(["(%s,%s)" % (x,y) for (x,y) in points])
    interior = IFELSE(background, "fill=%s" % background, '')
    boundary = ''
    if linewidth or linecolor:
        if not linecolor: linecolor='black'
        if not linewidth: linewidth=0.02
        boundary = 'draw=%s, line width=%scm' % (linecolor, linewidth)
    return r"\path [%s, %s] %s;" % (interior, boundary, points_str)

p = Plot()
points = [(0,2)] +\
         [(i/2.0,random.randrange(3*2,5*2)/4.0) for i in range(1, 5)] +\
         [(i/2.0,random.randrange(0,3*2)/4.0) for i in range(5,1,-1)]
p += polygon(points=points, background='blue!20')
p += polygon(points=[(x+4,y) for (x,y) in points], linewidth=0.02)
p += polygon(points=[(x+8,y) for (x,y) in points], linewidth=0.02,
             background='blue!20')
p += polygon(points=[(x+12,y) for (x,y) in points], linewidth=0.1,
             background='blue!20', linecolor='red')
print(p)
```



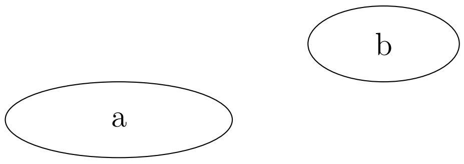
debug: relati
placement.te

23 Relative node placement

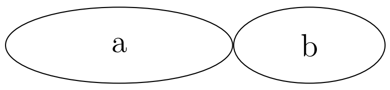
Temporary fix to relative node placement: use the `next_to` function.

http://tex.stackexchange.com/questions/69439/how-can-i-achieve-relative-positioning-i

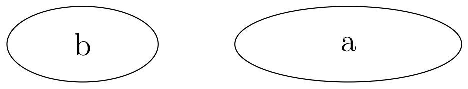
```
from latextool_basic import *
p = Plot()
p += ellipse(1, 1, 4, 2, name='a', label='a')
b = str(ellipse(5, 2, 7, 3, name='b', label='b'))
p += b
print(p)
```



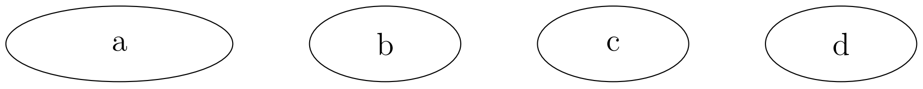
```
from latextool_basic import *
p = Plot()
p += ellipse(1, 1, 4, 2, name='a', label='a')
b = str(ellipse(5, 2, 7, 3, name='b', label='b'))
b = next_to(b, name='a', directions=('right', 0))
p += b
print(p)
```



```
from latextool_basic import *
p = Plot()
p += ellipse(1, 1, 4, 2, name='a', label='a')
b = str(ellipse(5, 2, 7, 3, name='b', label='b'))
b = next_to(b, name='a', directions=('left', 1))
p += b
print(p)
```

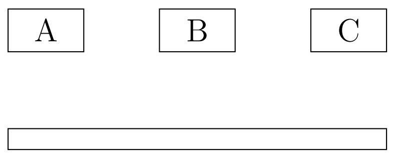


```
from latextool_basic import *
p = Plot()
p += ellipse(1, 1, 4, 2, name='a', label='a')

x = str(ellipse(5, 2, 7, 3, name='b', label='b'))
x = next_to(x, name='a', directions=('right', 1))
p += x

x = str(ellipse(5, 2, 7, 3, name='c', label='c'))
x = next_to(x, name='b', directions=('right', 1))
p += x

x = str(ellipse(5, 2, 7, 3, name='d', label='d'))
x = next_to(x, name='c', directions=('right', 1))
p += x

print(p)
```



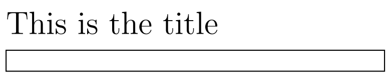
Example that uses both =of and anchor:

```
from latextool_basic import *
p = Plot()
p += r'\node[minimum width=5cm, draw, rectangle] (basis) {};'
p += r'\node[above=of basis.north west, anchor=south west, draw, rectangle,
 minimum width=1cm] (a) {A};'
p += r'\node[above=of basis, draw, rectangle,minimum width=1cm] (b) {B};'
p += r'\node[above=of basis.north east, anchor=south east, draw, rectangle,
 minimum width=1cm] (c) {C};'
print(p)
```



Example that uses both =of and anchor and labeling a rect. Basically flushing left on two nodes.

```
from latextool_basic import *
p = Plot()
p += r'\node[minimum width=5cm, draw, rectangle] (basis) {};'
p += r'\node[above=0.2cm of basis.north west, anchor=south west, rectangle,
 minimum width=1cm, inner sep=0cm] (a) {This is the title};'
print(p)
```

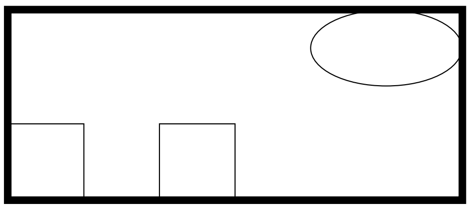


debug:
boundingbox.

24 Bounding nodes

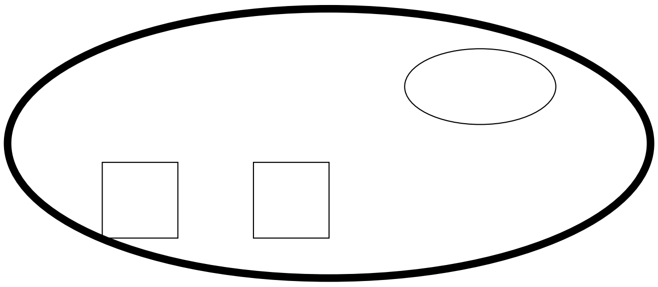
Bounding box (rectangle) for a group of tikz nodes:

```
from latextool_basic import *
p = Plot()
p += Rect(0,0,1,1,name='a')
p += Rect(2,0,3,1,name='b')
p += ellipse(center=(5,2), width=2, height=1, name='c')
p += r'\node [draw=black, fit=(a) (b) (c), line width=0.1cm, inner sep=0.0c
m] {};'
print(p)
```



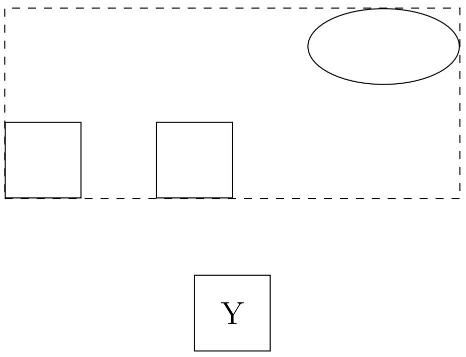
Bounding ellipse for a group of tikz nodes:

```
from latextool_basic import *
p = Plot()
p += Rect(0,0,1,1,name='a')
p += Rect(2,0,3,1,name='b')
p += ellipse(center=(5,2), width=2, height=1, name='c')
p += r'\node [ellipse, draw=black, fit=(a) (b) (c), line width=0.1cm, inner
 sep=0.0cm] {};'
print(p)
```



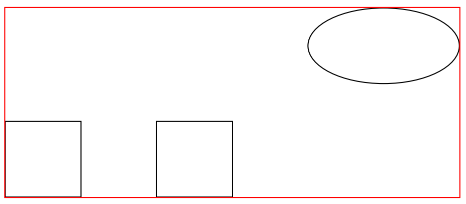
Put a node below a group of nodes:

```
from latextool_basic import *
p = Plot()
p += Rect(0,0,1,1,name='a')
p += Rect(2,0,3,1,name='b')
p += ellipse(center=(5,2), width=2, height=1, name='c')
p += r'\node [draw=black, fit=(a) (b) (c), inner sep=0.0cm, dashed] (X) {};
'
p += r'\node [draw=black, rectangle, minimum width=1cm, below=1cm  of X, mi
nimum height=1cm, distance=2cm] (Y) {Y};'
print(p)
```



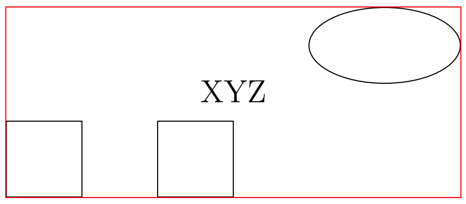
fit function:

```
from latextool_basic import *
p = Plot()
p += Rect(0,0,1,1,name='a')
p += Rect(2,0,3,1,name='b')
p += ellipse(center=(5,2), width=2, height=1, name='c')
p += fit(name='X', names=['a', 'b', 'c'], linecolor='red')
print(p)
```



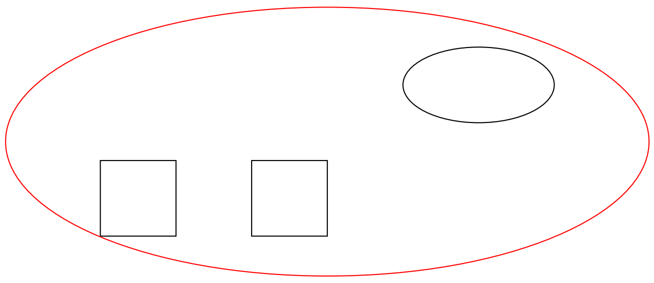
fit function with label:

```
from latextool_basic import *
p = Plot()
p += Rect(0,0,1,1,name='a')
p += Rect(2,0,3,1,name='b')
p += ellipse(center=(5,2), width=2, height=1, name='c')
p += fit(name='X', names=['a', 'b', 'c'], linecolor='red', label='XYZ')
print(p)
```



fit function with shape:

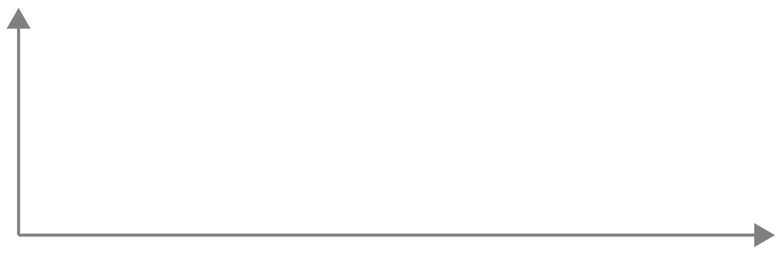
```
from latextool_basic import *
p = Plot()
p += Rect(0,0,1,1,name='a')
p += Rect(2,0,3,1,name='b')
p += ellipse(center=(5,2), width=2, height=1, name='c')
p += fit(name='X', names=['a', 'b', 'c'], shape='ellipse', linecolor='red')
print(p)
```



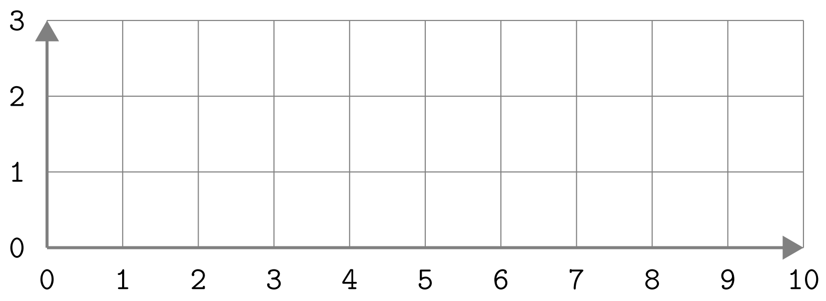
debug: axes.t

25 Axes

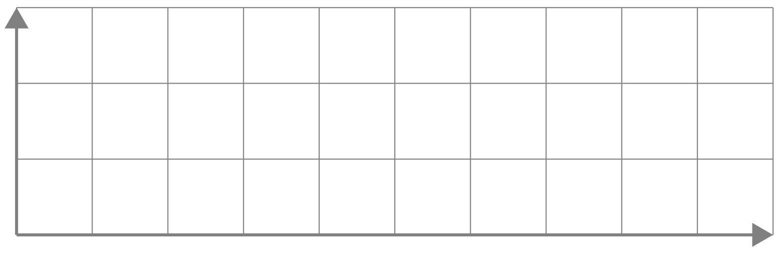
```
from latextool_basic import *
p = Plot()
axes(p, x0=0, y0=0, x1=10, y1=3)
print(p)
```



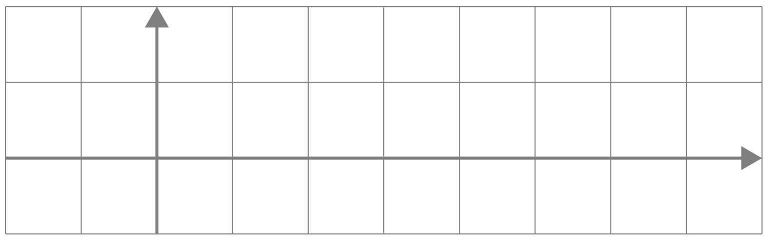
```
from latextool_basic import *
p = Plot()
p += Grid(x0=0, y0=0, x1=10, y1=3)
axes(p, x0=0, y0=0, x1=10, y1=3)
print(p)
```



```
from latextool_basic import *
p = Plot()
p += Grid(x0=0, y0=0, x1=10, y1=3, label_axes=False)
axes(p, x0=0, y0=0, x1=10, y1=3)
print(p)
```

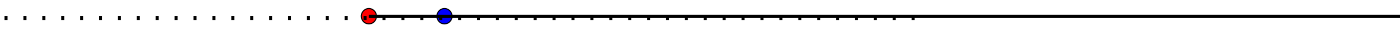


```
from latextool_basic import *
p = Plot()
p += Grid(x0=-2, y0=-1, x1=8, y1=2, label_axes=False)
axes(p, x0=-2, y0=-1, x1=8, y1=2)
print(p)
```



NEW VERSION

```
from latextool_basic import *
p = Plot()
xaxis = XAxis(x0=0, y0=0,
              start=5.0, end=20.0,
              xscale=1.0,
              start_tick=0.2, end_tick=12.25,
              tick_gap=0.25,
              tick_len=0.05,
              start_label=6.0, end_label=18.0,
              label_gap=1,
              label_anchor_gap=0,
              label_fontsize=r'\small',
              arrowhead=None)

p += Circle(x=0, y=0, r=0.1, background='red')
p += Circle(x=1, y=0, r=0.1, background='blue')
p += str(xaxis)
print(p)
```



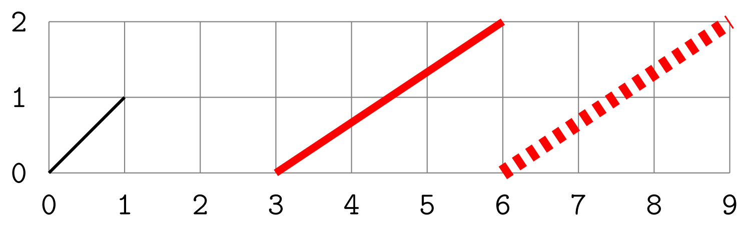
debug: chap-

26 Lines

debug: line.te

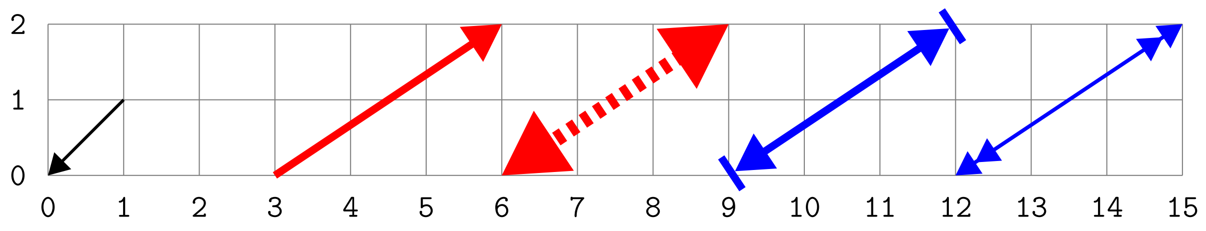
26.1 linecolor and linewidth

```
from latextool_basic import *
p = Plot()
p += Line(x0=0, y0=0, x1=1, y1=1)
p += Line(x0=3, y0=0, x1=6, y1=2, linecolor='red', linewidth=0.1)
p += Line(x0=6, y0=0, x1=9, y1=2, linecolor='red', linewidth=0.2,
                                  linestyle='dashed')
p += Grid()
print(p)
```



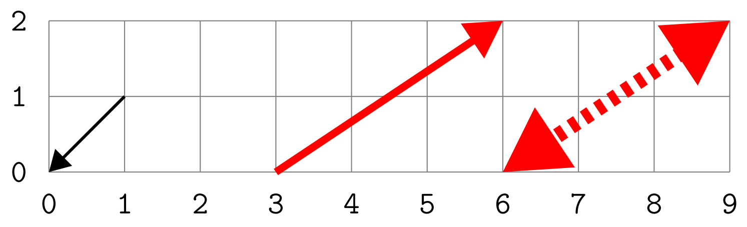
26.2 startstyle and endstyle

```
from latextool_basic import *
p = Plot()
p += Line(x0=0, y0=0, x1=1, y1=1, startstyle='>')
p += Line(x0=3, y0=0, x1=6, y1=2, linecolor='red', linewidth=0.1,
                                  endstyle='>')
p += Line(x0=6, y0=0, x1=9, y1=2, linecolor='red', linewidth=0.2,
                                  linestyle='dashed',
                                  startstyle='>', endstyle='>')
p += Line(x0=9, y0=0, x1=12, y1=2, linecolor='blue', linewidth=0.1,
                                   startstyle='>|', endstyle='>|')
p += Line(x0=12, y0=0, x1=15, y1=2, linecolor='blue', linewidth=0.05,
                                   startstyle='>>', endstyle='>>')
p += Grid()
print(p)
```



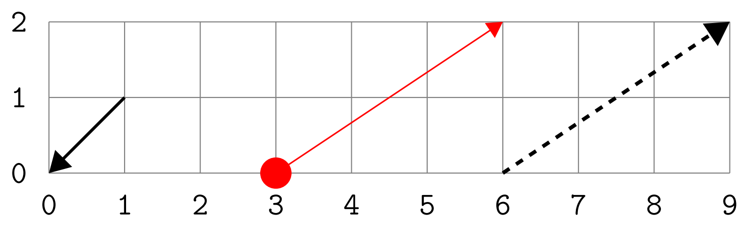
`arrowstyle=triangle`.

```
from latextool_basic import *
p = Plot()
p += Line(x0=0, y0=0, x1=1, y1=1, startstyle='->', arrowstyle='triangle')
p += Line(x0=3, y0=0, x1=6, y1=2, linecolor='red', linewidth=0.1,
                                  endstyle='->', arrowstyle='triangle')
p += Line(x0=6, y0=0, x1=9, y1=2, linecolor='red', linewidth=0.2,
                                  linestyle='dashed',
                                  startstyle='->', endstyle='->',
                                  arrowstyle='triangle')
p += Grid()
print(p)
```



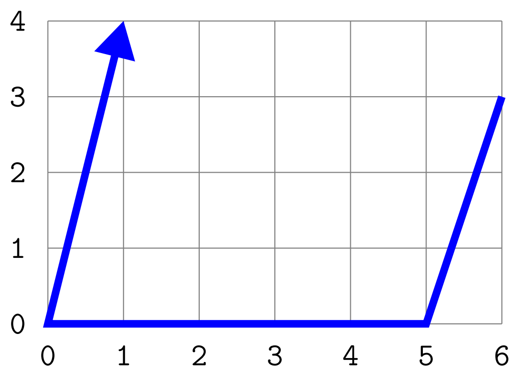
startstyle='dot'.

```
from latextool_basic import *
p = Plot()
p += Line(x0=0, y0=0, x1=1, y1=1, startstyle='->',
                                  arrowstyle='triangle', endstyle='dot')
p += Line(x0=3, y0=0, x1=6, y1=2, linecolor='red', linewidth=0.02,
                                  endstyle='->', arrowstyle='triangle',
                                  startstyle='dot', r=0.2)
p += Line(x0=6, y0=0, x1=9, y1=2, linecolor='black', linewidth=0.05,
                                  linestyle='dashed', startstyle='dot',
                                  endstyle='->',  arrowstyle='triangle')
p += Grid()
print(p)
```



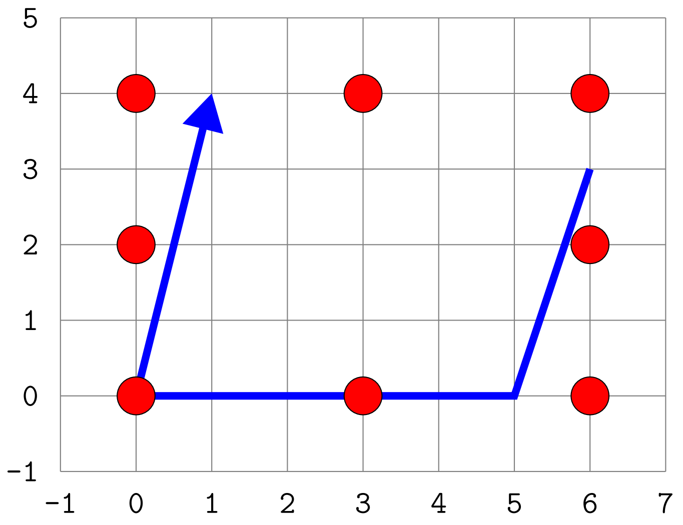
26.3 points parameter

```
from latextool_basic import *
p = Plot()
p += Line(points=[(1,4), (0,0), (5,0), (6,3)],
          linecolor='blue', linewidth=0.1,
          startstyle='->', arrowstyle='triangle', endstyle='dot')
p += Grid()
print(p)
```



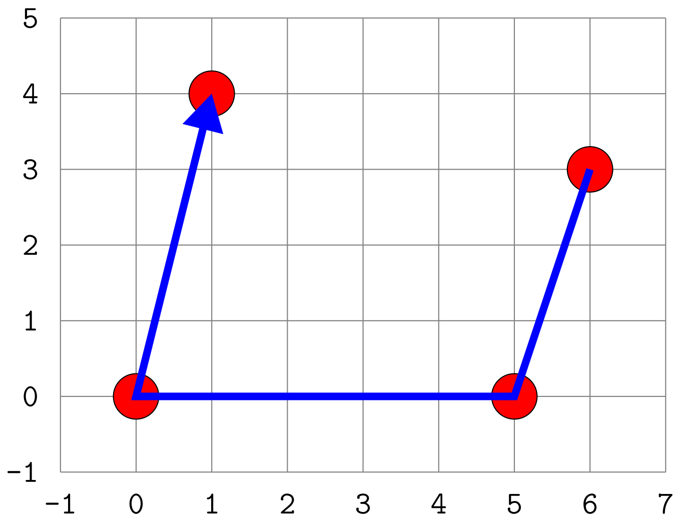
26.4 boundary points

```
from latextool_basic import *
p = Plot()

aline = Line(points=[(1,4), (0,0), (5,0), (6,3)],
             linecolor='blue', linewidth=0.1,
             startstyle='->', arrowstyle='triangle', endstyle='dot')
p += aline
p += Circle(center=aline.top(), r=0.25, background='red')
p += Circle(center=aline.bottom(), r=0.25, background='red')
p += Circle(center=aline.left(), r=0.25, background='red')
p += Circle(center=aline.right(), r=0.25, background='red')
p += Circle(center=aline.topleft(), r=0.25, background='red')
p += Circle(center=aline.topright(), r=0.25, background='red')
p += Circle(center=aline.bottomleft(), r=0.25, background='red')
p += Circle(center=aline.bottomright(), r=0.25, background='red')
p += Grid()
print(p)
```



Test points

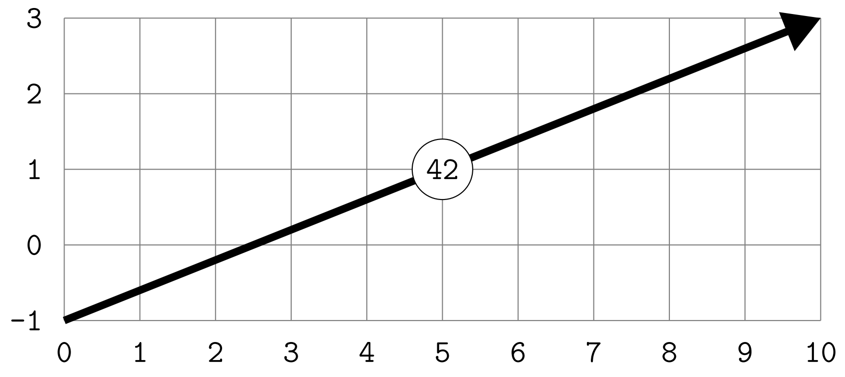
```
from latextool_basic import *
p = Plot()

aline = Line(points=[(1,4), (0,0), (5,0), (6,3)],
             linecolor='blue', linewidth=0.1,
             startstyle='->', arrowstyle='triangle', endstyle='dot')
for point in aline.points:
    p += Circle(center=point, r=0.3, background='red')

p += aline
p += Grid()
print(p)
```



26.5 Midpoints

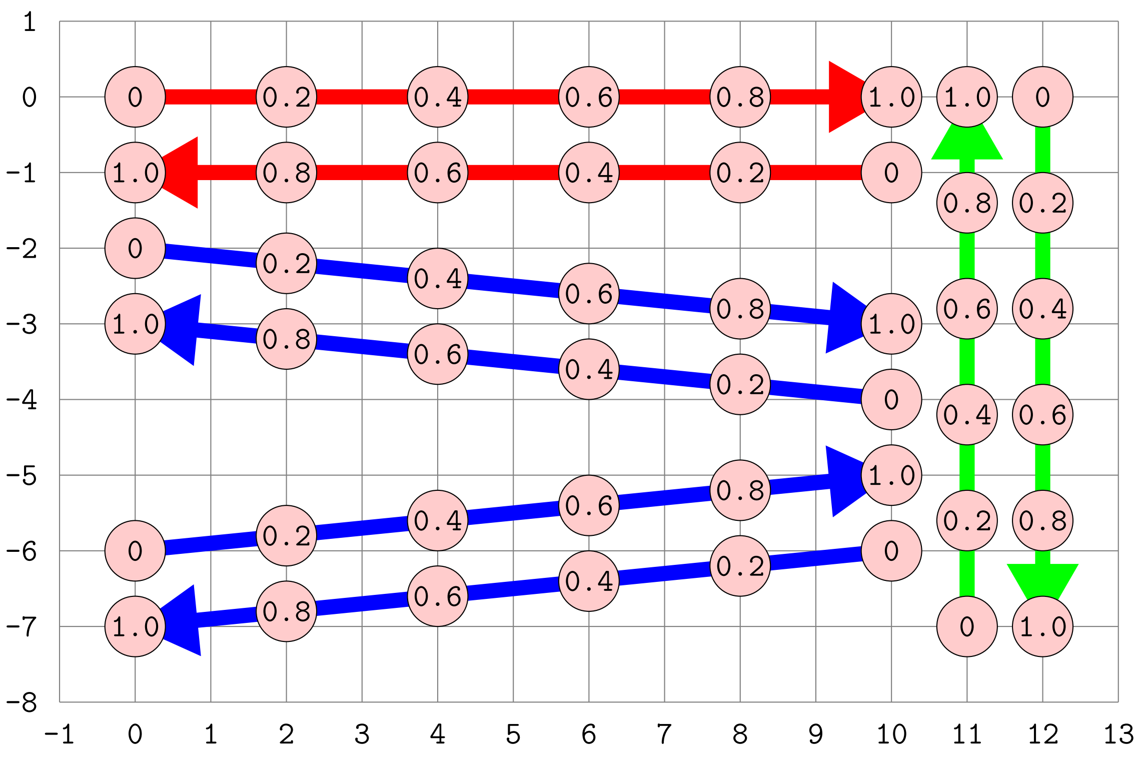
Test midpoint (can be useful for weighted graphs and network flows.

```
from latextool_basic import *
p = Plot()
points = [(0,-1), (10,3)]
aline = Line(points=points, endstyle='>', linewidth=0.1)
p += aline

p += Circle(center=aline.midpoint(), r=0.4,
            background='white', label=r'{\texttt{%s}}' % 42)
p += Grid()
print(p)
```



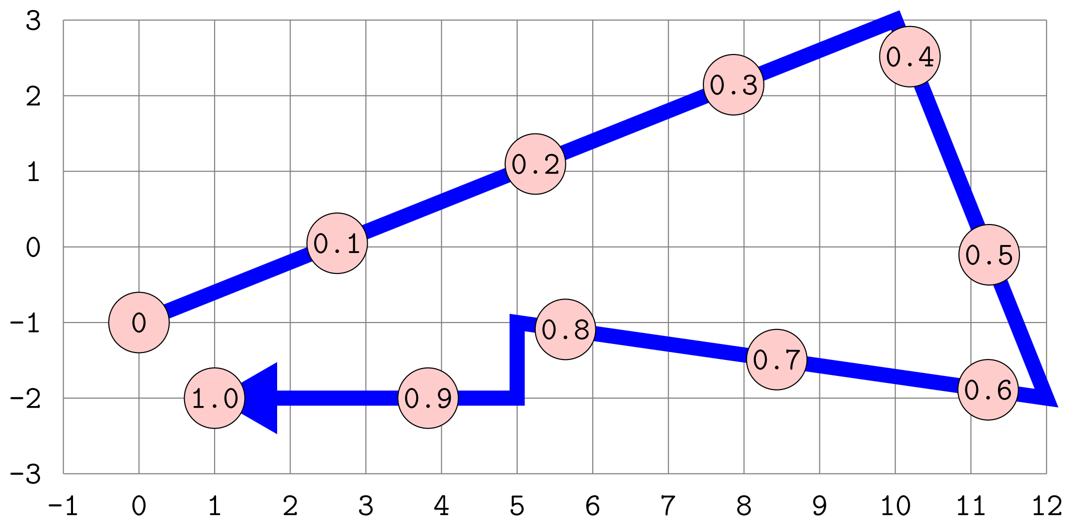
Test midpoint with different ratios.

```
from latextool_basic import *
p = Plot()
def linewithpoints(p, points, color1, color2):
    aline = Line(points=points, linecolor=color1, linewidth=0.2,
                 startstyle='dot', arrowstyle='triangle', endstyle='->')
    p += aline
    for i in [0, 0.2, 0.4, 0.6, 0.8, 1.0]:
        p += Circle(center=aline.midpoint(ratio=i), r=0.4,
                    background=color2, label=r'{\texttt{%s}}' % i)
linewithpoints(p, [(0, 0), (10, 0)], 'red', 'red!20')
linewithpoints(p, [(10, -1), (0, -1)], 'red', 'red!20')
linewithpoints(p, [(0, -2), (10, -3)], 'blue', 'red!20')
linewithpoints(p, [(10, -4), (0, -3)], 'blue', 'red!20')
linewithpoints(p, [(0, -6), (10, -5)], 'blue', 'red!20')
linewithpoints(p, [(10, -6), (0, -7)], 'blue', 'red!20')
linewithpoints(p, [(11, -7), (11, 0)], 'green', 'red!20')
linewithpoints(p, [(12, 0), (12, -7)], 'green', 'red!20')
p += Grid()
print(p)
```



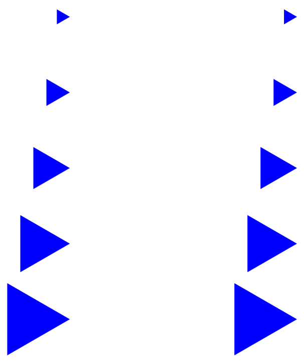
Test midpoint for 1 segment.

```
from latextool_basic import *
p = Plot()
points = [(0,-1), (10,3), (12,-2), (5,-1), (5,-2), (1,-2)]
aline = Line(points=points, linecolor='blue', linewidth=0.2,
             startstyle='dot', arrowstyle='triangle', endstyle='->')
p += aline
for i in [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]:
    p += Circle(center=aline.midpoint(ratio=i), r=0.4,
                background='red!20', label=r'{\texttt{%s}}' % i)
p += Grid()
print(p)
```



Test short line with arrow tip (basically to test arrow tip).

```
from latextool_basic import *
p = Plot()
for i, x in enumerate([0.2, 0.15, 0.1, 0.05, 0.01]):
    p += Line(points=[(1-x, i), (1, i)], linecolor='blue', linewidth=x,
              endstyle='->')
    p += Line(points=[(4-x, i), (4, i)], linecolor='blue', linewidth=x,
              endstyle='->', arrowstyle='triangle')
#p += Grid()
print(p)
```

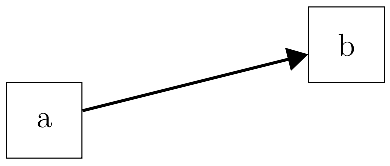


For arrows with line width x, it seems enough to have a length of x.

26.6 tikz names

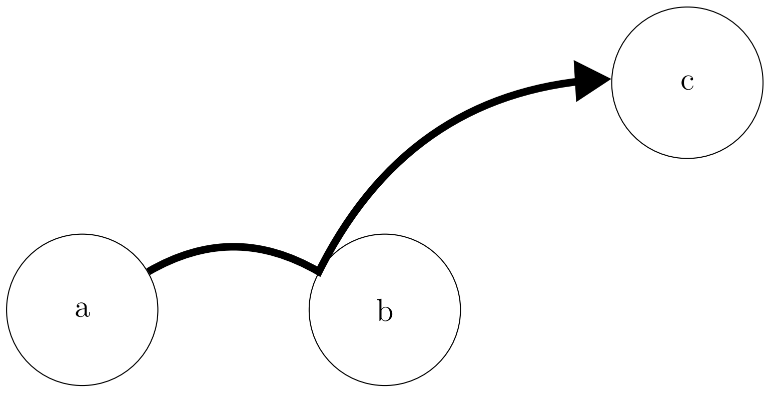
line from tikz name to tikz name:

```
from latextool_basic import *
p = Plot()
p += Rect(x0=1, y0=0, x1=2, y1=1, label='a', name='a')
p += Rect(x0=5, y0=1, x1=6, y1=2, label='b', name='b')

p += Line(names=['a', 'b'], endstyle='>')

print(p)
```



26.7 bend

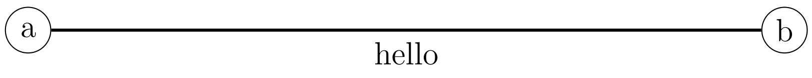
```
from latextool_basic import *
p = Plot()

p += Circle(x=0, y=0, r=1, name='a', label='a')
p += Circle(x=4, y=0, r=1, name='b', label='b')
p += Circle(x=8, y=3, r=1, name='c', label='c')

p += Line(names=['a','b','c'], bend_left=30, endstyle='>', linewidth=0.1)

print(p)
```



26.8 label and anchor

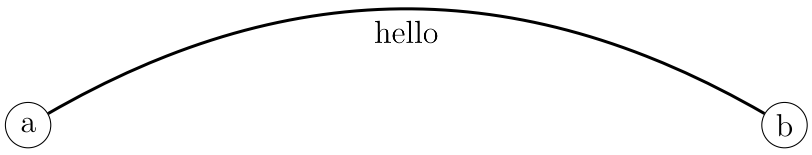
anchor = below

```
from latextool_basic import *
p = Plot()
p += Circle(x=0, y=0, r=0.3, name='a', label='a')
p += Circle(x=10, y=0, r=0.3, name='b', label='b')
p += Line(names=['a', 'b'], anchor='below', label='hello')
print(p)
```



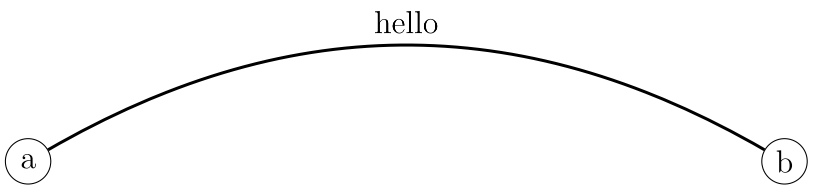
anchor = below with bend

```
from latextool_basic import *
p = Plot()
p += Circle(x=0, y=0, r=0.3, name='a', label='a')
p += Circle(x=10, y=0, r=0.3, name='b', label='b')
p += Line(names=['a', 'b'], bend_left=30, anchor='below', label='hello')
print(p)
```



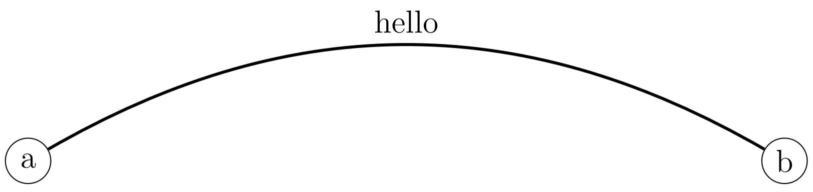
anchor = above

```
from latextool_basic import *
p = Plot()
p += Circle(x=0, y=0, r=0.3, name='a', label='a')
p += Circle(x=10, y=0, r=0.3, name='b', label='b')
p += Line(names=['a', 'b'], bend_left=30, anchor='above', label='hello')
print(p)
```



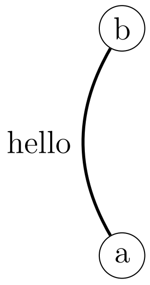
anchor = above with bend

```
from latextool_basic import *
p = Plot()
p += Circle(x=0, y=0, r=0.3, name='a', label='a')
p += Circle(x=10, y=0, r=0.3, name='b', label='b')
p += Line(names=['a', 'b'], bend_left=30, anchor='above', label='hello')
print(p)
```



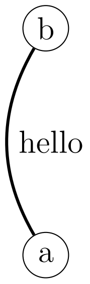
anchor = left with bend

```
from latextool_basic import *
p = Plot()
p += Circle(x=0, y=0, r=0.3, name='a', label='a')
p += Circle(x=0, y=3, r=0.3, name='b', label='b')
p += Line(names=['a', 'b'], bend_left=30, anchor='left', label='hello')
print(p)
```



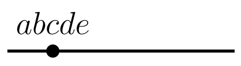
anchor = right with bend

```
from latextool_basic import *
p = Plot()
p += Circle(x=0, y=0, r=0.3, name='a', label='a')
p += Circle(x=0, y=3, r=0.3, name='b', label='b')
p += Line(names=['a', 'b'], bend_left=30, anchor='right', label='hello')
print(p)
```



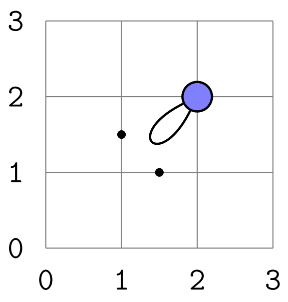
26.9 Using midpoint to place labels

```
from latextool_basic import *
p = Plot()
aline = Line(points=[(0,0), (3,0)])
p += aline
x,y = aline.midpoint(ratio=0.2)

from latexcircuit import *
p += Circle(x=x, y=y, r=0.05, background='red')
X = POINT(x=x, y=y, label='$abcde$', anchor='south')
p += str(X)

print(p)
```



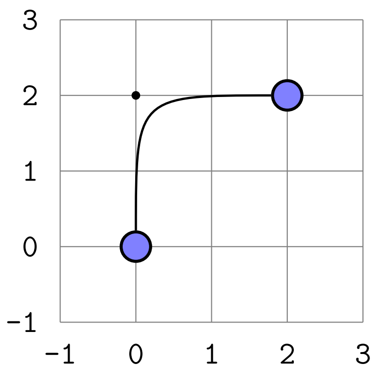
26.10 Two control points

```
from latextool_basic import *
p = Plot()
L = 0.03
C = 'blue!50'

p += Circle(x=2, y=2, r=0.2, background=C, name='a', linewidth=L)
p += Circle(x=1, y=1.5, r=0.05, background='black')
p += Circle(x=1.5, y=1, r=0.05, background='black')

p += Line(names=['a','a'], controls=[(1,1.5),(1.5,1)], linewidth=L)
p += Grid(0, 0, 3, 3)
print(p)
```



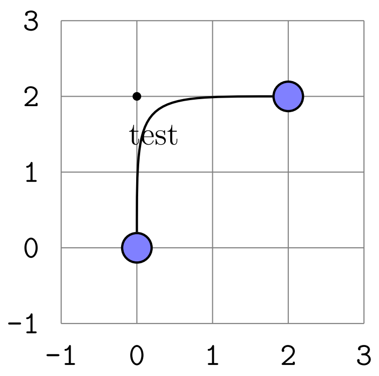
26.11 One control point

```
from latextool_basic import *
p = Plot()
C = 'blue!50'

p += Circle(x=2, y=2, r=0.2, background=C, name='22', linewidth=L)
p += Circle(x=0, y=0, r=0.2, background=C, name='00', linewidth=L)

p += Circle(x=0, y=2, r=0.05, background='black')

p += Line(names=['22','00'], controls=[(0,2)], linewidth=0.03)
p += Grid(x0=-1,y0=-1,x1=3,y1=3)

print(p)
```



1 control point and label

```
from latextool_basic import *
p = Plot()
L = 0.03
C = 'blue!50'

p += Circle(x=2, y=2, r=0.2, background=C, name='22', linewidth=L)
p += Circle(x=0, y=0, r=0.2, background=C, name='00', linewidth=L)

p += Circle(x=0, y=2, r=0.05, background='black')

p += Line(names=['22','00'], controls=[(0,2)], linewidth=L, label='test', a
nchor='below')

p += Grid(x0=-1,y0=-1,x1=3,y1=3)
print(p)
```

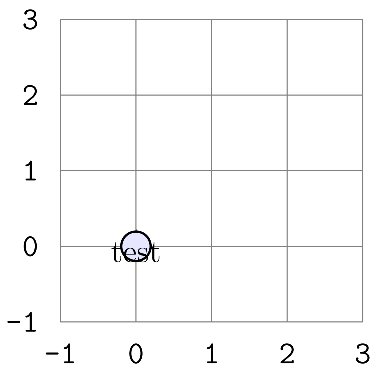


26.12 loop

Example uses Graph.

xxx

```
from latextool_basic import *
p = Plot()
L = 0.03
C = 'blue!50'

p += Graph.node(x=0, y=0, name='a')
p += Line(names=['a','a'], label='test', anchor='below', loop='loop above')

p += Grid(x0=-1,y0=-1,x1=3,y1=3)
print(p)
```

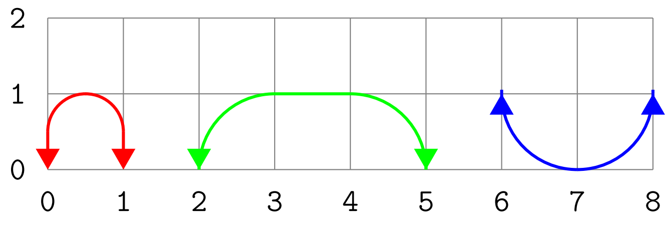

3
2
1
0
-1
-1
0
1
2
3
test

debug: bend.

26.13 bend

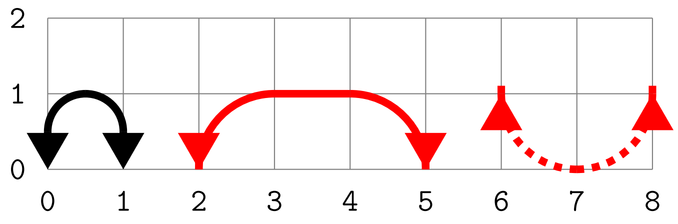
Basically used for drawing swaps in array sorting.

```
from latextool_basic import Plot, Grid, bend
p = Plot()
p += bend([0,0], [1, 0], dy=1, linecolor='red')
p += bend([2,0], [5, 0], dy=1, linecolor='green')
p += bend([6,1], [8, 1], dy=-1, linecolor='blue')
p += Grid(x0=0, y0=0, x1=8, y1=2)
print(p)
```



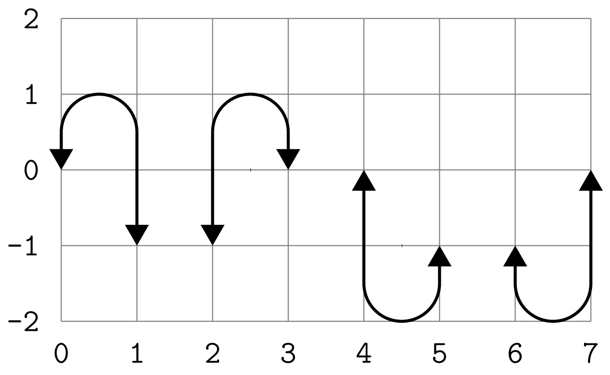
You can specify linewdith, linecolor, linestyle:

```
from latextool_basic import Plot, Grid, bend
p = Plot()
p += bend([0,0], [1, 0], dy=1, linewidth=0.1)
p += bend([2,0], [5, 0], dy=1, linewidth=0.1, linecolor='red')
p += bend([6,1], [8, 1], dy=-1, linewidth=0.1, linecolor='red',
          linestyle='dashed')
p += Grid(x0=0, y0=0, x1=8, y1=2)
print(p)
```



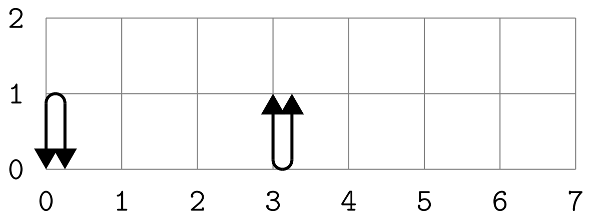
The y values can be different:

```
from latextool_basic import Plot, Grid, bend
p = Plot()
p += bend([0,0], [1, -1], dy=1)
p += bend([2,-1], [3, 0], dy=1)
p += bend([4,0], [5, -1], dy=-1)
p += bend([6,-1], [7, 0], dy=-1)
p += Grid(x0=0, y0=-2, x1=7, y1=2)
print(p)
```



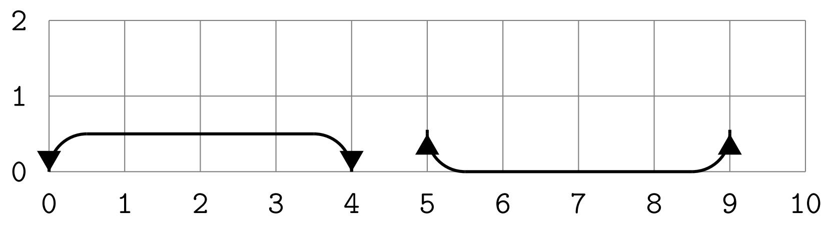
default radius (dy too large):

```
from latextool_basic import Plot, Grid, bend
p = Plot()
s0 = bend([0,0], [0.25,0], dy=1)
s1 = bend([3,1], [3.25,1], dy=-1)
p.add(s0)
p.add(s1)
p += Grid(x0=0, y0=0, x1=7, y1=2)
print(p)
```



default radius (dx too large):

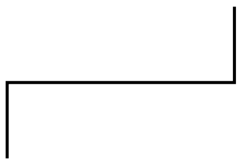
```
from latextool_basic import Plot, Grid, bend
p = Plot()
s0 = bend([0,0], [4,0], dy=0.5)
s1 = bend([5,0.5], [9,0.5], dy=-0.5)
p.add(s0)
p.add(s1)
p += Grid(x0=0, y0=0, x1=10, y1=2)
print(p)
```



debug:
orthogonal-li

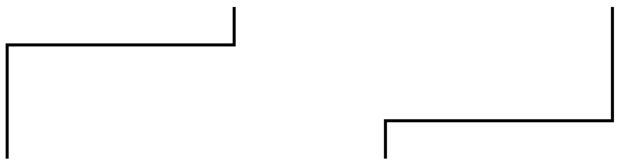
26.14 Orthogonal lines; organization chart

```
from latextool_basic import *
p = Plot()
points = get_points(0, 0, 3, 2, 'vbroom')
p += Line(points = points)
print(p)
```

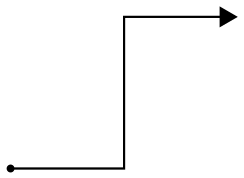


With adjustment:

```
from latextool_basic import *
p = Plot()

x = 0
points = get_points(x, 0, x+3, 2, 'vbroom', delta=0.5)
p += Line(points = points)

x = 5
points = get_points(x, 0, x+3, 2, 'vbroom', delta=-0.5)
p += Line(points = points)

print(p)
```

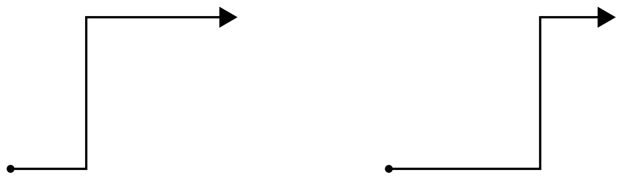


```
from latextool_basic import *
p = Plot()
points = get_points(0, 0, 3, 2, 'hbroom')
p += Pointer(points = points)
print(p)
```

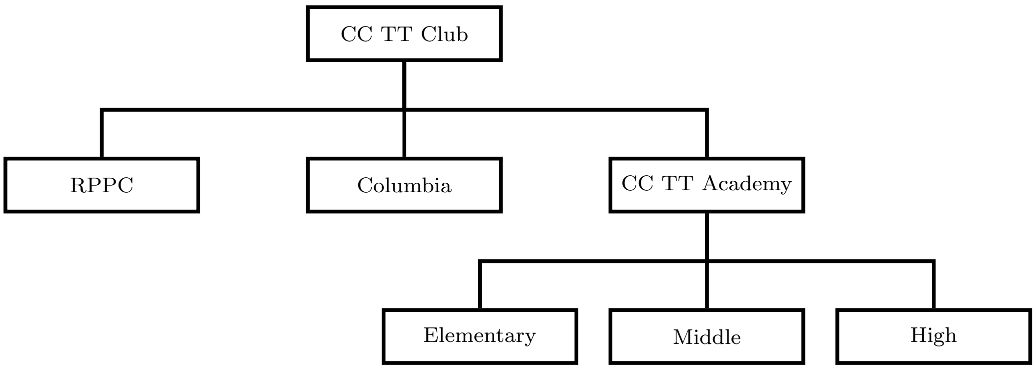


With adjustment:

```
from latextool_basic import *
p = Plot()

x = 0
points = get_points(x, 0, x+3, 2, 'hbroom', delta=-0.5)
p += Pointer(points = points)

x = 5
points = get_points(x, 0, x+3, 2, 'hbroom', delta=+0.5)
p += Pointer(points = points)

print(p)
```



```
from latextool_basic import *
p = Plot()
d = positions(r'''
    A

B   C   D

     E  F  G
''')
label = {'A':'CC TT Club',    'B':'RPPC',       'C':'Columbia',
         'D':'CC TT Academy', 'E':'Elementary', 'F':'Middle', 'G':'High'}
WIDTH = 2.6; HEIGHT = 0.5
rect = {}
for lab, pos in d.items():
    x, y = pos
    rect[lab] =  Rect(x0=x-WIDTH/2, y0=y-HEIGHT/2,
                 x1=x+WIDTH/2, y1=y+HEIGHT, label=r'{\scriptsize{%s}}' % la
bel[lab],
                 linewidth=0.05)
    p += rect[lab]

for lab0, lab1s in [('A', ('B','C','D')),
                    ('D', ('E','F','G')),
                    ]:
    x0,y0 = rect[lab0].bottom()
    for lab1 in lab1s:
        x1,y1 = rect[lab1].top()
        points = get_points(x0,y0,x1,y1,'vbroom')
        p += Line(points=points, linewidth=0.05)

print(p)
```



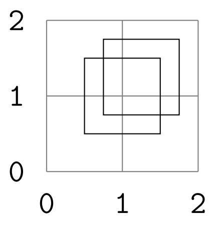
debug: rect.t

27 Rect class

27.1 No background

TEST: Background is '', i.e., totally transparent.

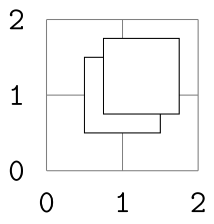
```
from latextool_basic import *
p = Plot()
p += Rect(x0=0.5, y0=0.5, x1=1.5, y1=1.5)
p += Rect(x0=0.75, y0=0.75, x1=1.75, y1=1.75)
p += Grid()
print(p)
```



27.2 Background

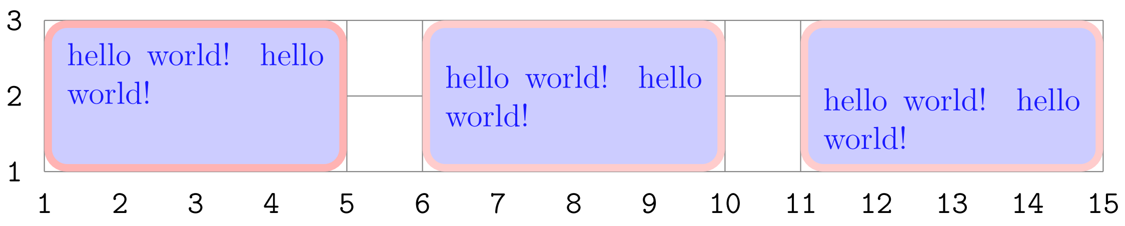
TEST: Background is white

```
from latextool_basic import *
p = Plot()
p += Rect(x0=0.5, y0=0.5, x1=1.5, y1=1.5, background='white')
p += Rect(x0=0.75, y0=0.75, x1=1.75, y1=1.75, background='white')
p += Grid()
print(p)
```



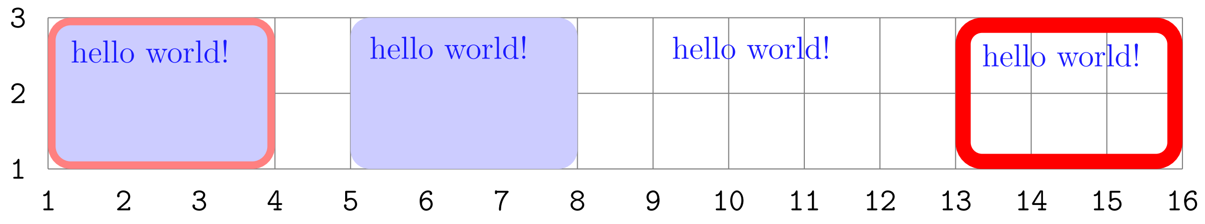
27.3 border, background, foreground

```
from latextool_basic import *
p = Plot()
p += Rect(x0=1, y0=1, x1=5, y1=3,
          radius=0.25, innersep=0.25,
          linecolor='red!30!white', linewidth=0.1,
          background='blue!20!white!', foreground='blue!90!white',
          s='hello world! hello world!', align='t')
p += Rect(x0=6, y0=1, x1=10, y1=3,
          radius=0.25, innersep=0.25,
          linecolor='red!20!white', linewidth=0.1,
          background='blue!20!white!', foreground='blue!90!white',
          s='hello world! hello world!', align='c')
p += Rect(x0=11, y0=1, x1=15, y1=3,
          radius=0.25, innersep=0.25,
          linecolor='red!20!white', linewidth=0.1,
          background='blue!20!white!', foreground='blue!90!white',
          s='hello world! hello world!', align='b')
p += Grid()
print(p)
```


hello world! hello world!
hello world! hello world!
hello world! hello world!
1
2
3
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

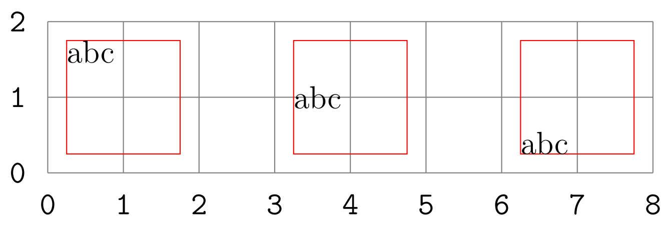
27.4 border, background, foreground

```
from latextool_basic import *
p = Plot(radius=0.25, innersep=0.25,
         linecolor='red!50!white', linewidth=0.1,
         background='blue!20!white', foreground='blue!90!white',
         align='t')
p += RectAdaptor(env=p.env, x0= 1, y0=1, x1= 4, y1=3, s='hello world!')
p.env['linewidth'] = 0
p += RectAdaptor(env=p.env, x0= 5, y0=1, x1= 8, y1=3, s='hello world!')
p.env['background'] = ''
p += RectAdaptor(env=p.env, x0= 9, y0=1, x1= 12, y1=3, s='hello world!')

p.env['background'] = ''
p.env['linecolor'] = 'red'
p.env['linewidth'] = 0.2
p += RectAdaptor(env=p.env, x0= 13, y0=1, x1= 16, y1=3, s='hello world!')

p += Grid()
print(p)
```



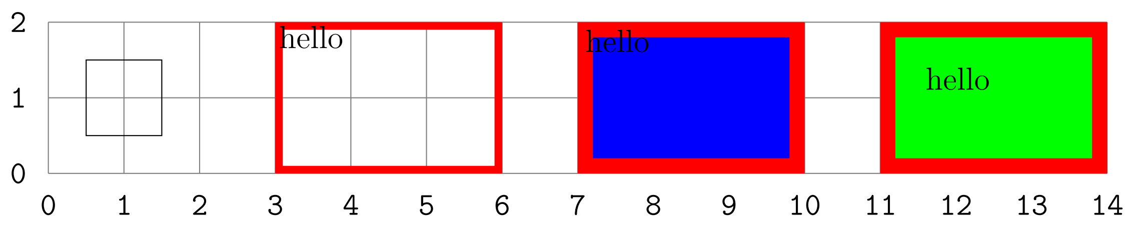
27.5 align

```
from latextool_basic import *
p = Plot()
p += Rect(x0=0.25, y0=0.25, x1=1.75, y1=1.75, s='abc', align='t',
          linecolor='red', foreground='black')
p += Rect(x0=3.25, y0=0.25, x1=4.75, y1=1.75, s='abc', align='c',
          linecolor='red')
p += Rect(x0=6.25, y0=0.25, x1=7.75, y1=1.75, s='abc', align='b',
          linecolor='red')
p += Grid()
print(p)
```



27.6 innersep

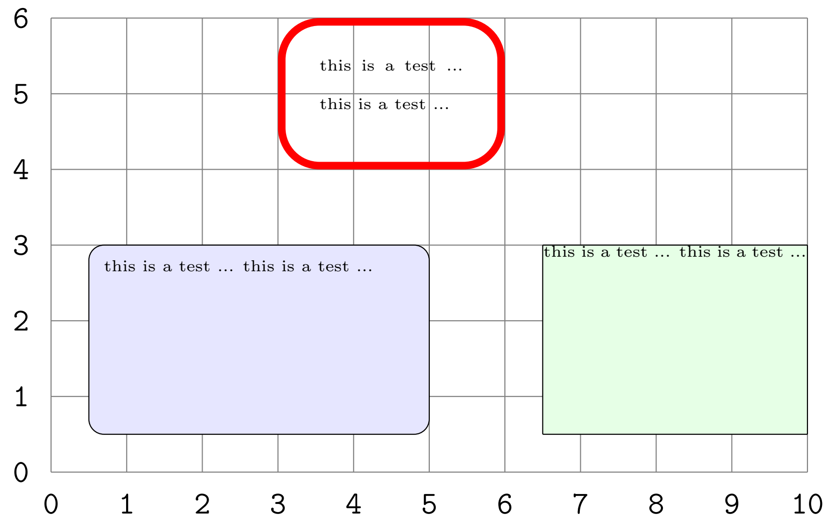
```
from latextool_basic import *
p = Plot()
p += Rect(x0=0.5, y0=0.5, x1=1.5, y1=1.5)
p += Rect(x0=3, y0=0, x1=6, y1=2, linecolor='red', linewidth=0.1, s='hello'
)
p += Rect(x0=7, y0=0, x1=10, y1=2, linecolor='red', background='blue',
          linewidth=0.2, s='hello')
p += Rect(x0=11, y0=0, x1=14, y1=2, linecolor='red', linewidth=0.2,
          background='green',
          innersep=0.5, s='hello')
p += Grid()
print(p)
```



27.7 rounded corners

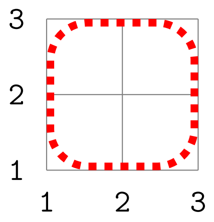
(radius of corner is set to innersep by default):

```
from latextool_basic import *
p = Plot()
p += Rect(x0=3, y0=4, x1=6, y1=6, linecolor='red', linewidth=0.1,
          innersep=0.5, radius=0.5,
          s=r'{\tiny this is a test ... this is a test ...}')
p += Rect(x0=0.5, y0=0.5, x1=5, y1=3,
          background='blue!10!white',
          innersep=0.2, radius=0.2,
          s=r'{\tiny this is a test ... this is a test ...}')
p += Rect(x0=6.5, y0=0.5, x1=10, y1=3,
          background='green!10!white',
          innersep=0.01, radius=0.01,
          s=r'{\tiny this is a test ... this is a test ...}')
p += Grid()
print(p)
```


this is a test ...
this is a test ...
this is a test ... this is a test ...
this is a test ... this is a test ...
0
1
2
3
4
5
6
7
8
9
10

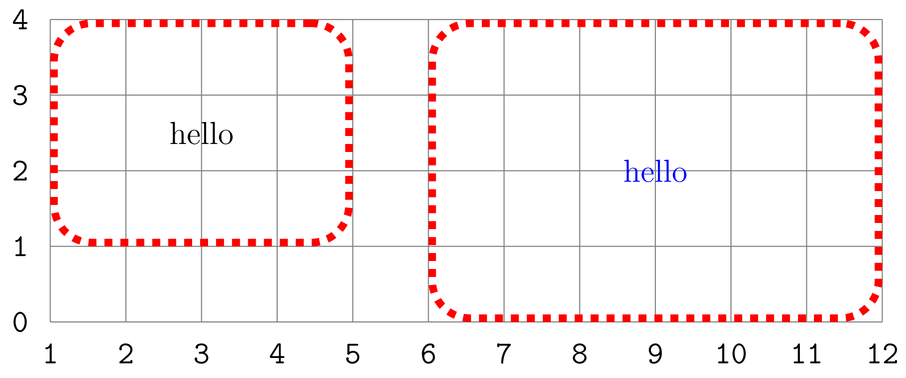
27.8 linestyle

```
from latextool_basic import *
p = Plot()
p += Rect(x0=1, y0=1, x1=3, y1=3,
          linewidth=0.1, linecolor='red',
          radius=0.5, linestyle='dashed')
p += Grid()
print(p)
```



27.9 label

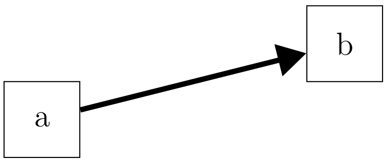
```
from latextool_basic import *
p = Plot()
p += Rect(x0=1, y0=1, x1=5, y1=4,
          linewidth=0.1, linecolor='red',
          radius=0.5, linestyle='dashed', label='hello')
p += Rect(x0=6, y0=0, x1=12, y1=4,
          linewidth=0.1, linecolor='red',
          foreground='blue',
          radius=0.5, linestyle='dashed', label='hello')
p += Grid()
print(p)
```



27.10 tikz name

Rect with tikz name:

```
from latextool_basic import *
p = Plot()
p += Rect(x0=1, y0=0, x1=2, y1=1, label='a', name='a')
p += Rect(x0=5, y0=1, x1=6, y1=2, label='b', name='b')

p += r"\draw[->,line width=2] (a) -- (b);"

print(p)
```

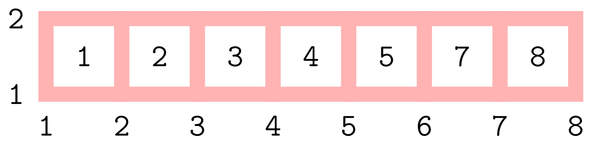


debug:
rectcontainer

28 RectContainer

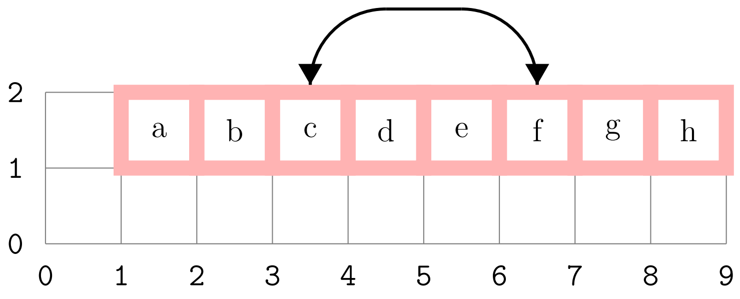
Horizontal array:

```
from latextool_basic import *
p = Plot()

c = RectContainer(x=1, y=1)
for x in '1234578':
    c += Rect2(x0=0, y0=0, x1=1, y1=1,
               linewidth=0.2, linecolor='red!30!white',
               label=r'{\texttt{%s}}' % x)
p += c
p += Grid()
print(p)
```



28.1 Using []

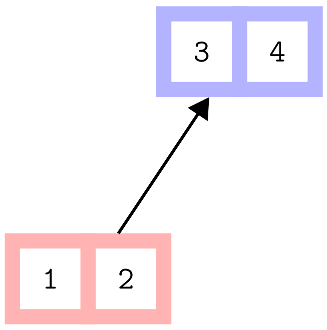
```
from latextool_basic import *
p = Plot()

c = RectContainer(x=1, y=1)
for x in 'abcdefgh':
    c += Rect2(x0=0, y0=0, x1=1, y1=1,
               linewidth=0.2, linecolor='red!30!white',
               label=x)
p += c
p += bend(c[2].top(), c[5].top())
p += Grid()
print(p)
```



tmp/68973ebe2b61653d167302cb3541f346.pdf

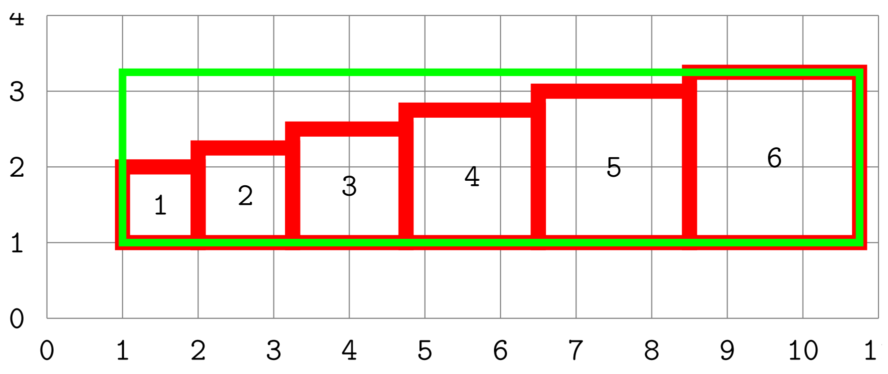
29 name

```
from latextool_basic import *
p = Plot()

c = RectContainer(x=1, y=1, name='A')
for x in '12':
    c += Rect2(x0=0, y0=0, x1=1, y1=1,
               linewidth=0.2, linecolor='red!30!white',
               label=r'{\texttt{%s}}' % x)

p += c

d = RectContainer(x=3, y=4, name='B')
for x in '34':
    d += Rect2(x0=0, y0=0, x1=1, y1=1,
               linewidth=0.2, linecolor='blue!30!white',
               label=r'{\texttt{%s}}' % x)

p += d

p += Line(names=['A','B'], endstyle='>')
print(p)
```



Left-to-right, align bottom:

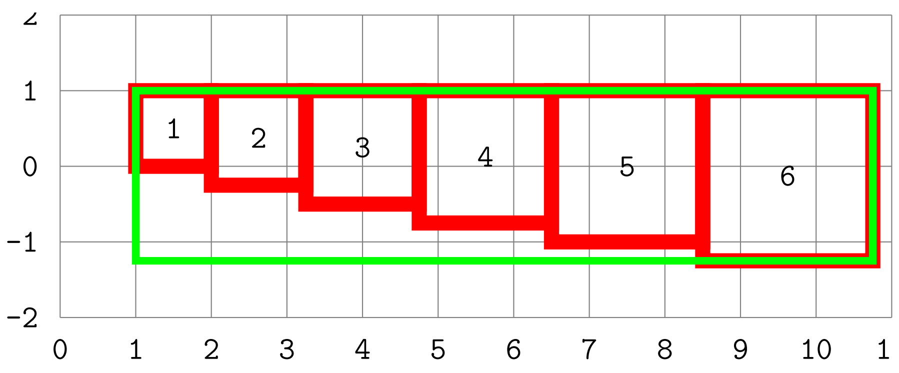
```
from latextool_basic import *
p = Plot()

c = RectContainer(x=1, y=1)
for i, x in enumerate([1,2,3,4,5,6]):
    c += Rect2(x0=0, y0=0, x1=1+i/4.0, y1=1+i/4.0, linecolor='red', linewid
th=0.2,
               label=r'{\texttt{%s}}' % x)
c.layout()
p += c

# boundary of container
p += Rect2(x0=c.x0, y0=c.y0, x1=c.x1, y1=c.y1, linewidth=0.1, linecolor='gr
een')

p += Grid()
print(p)
```



tmp/9fd9b7ae23523bc3e06b3bbc7b45a579.pdf

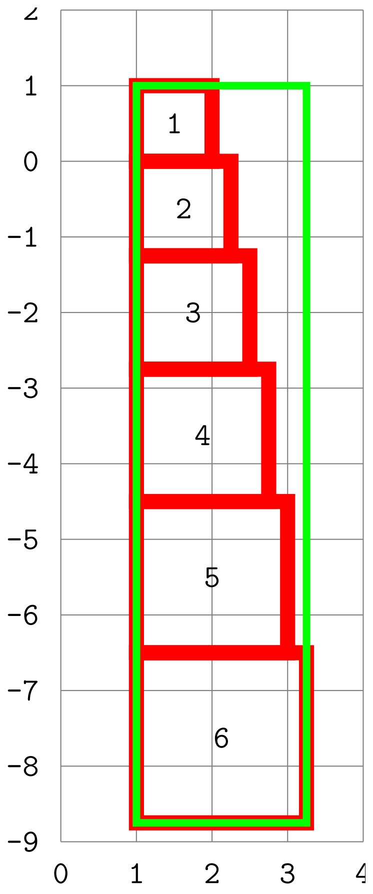
Align top, left-to-right:

```
from latextool_basic import *
p = Plot()

c = RectContainer(x=1, y=1, align='top')
for i, x in enumerate([1,2,3,4,5,6]):
    c += Rect2(x0=0, y0=0, x1=1+i/4.0, y1=1+i/4.0, linecolor='red', linewid
th=0.2,
               label=r'{\texttt{%s}}' % x)
p += c

# boundary of container
p += Rect2(x0=c.x0, y0=c.y0, x1=c.x1, y1=c.y1, linewidth=0.1, linecolor='gr
een')

p += Grid()
print(p)
```



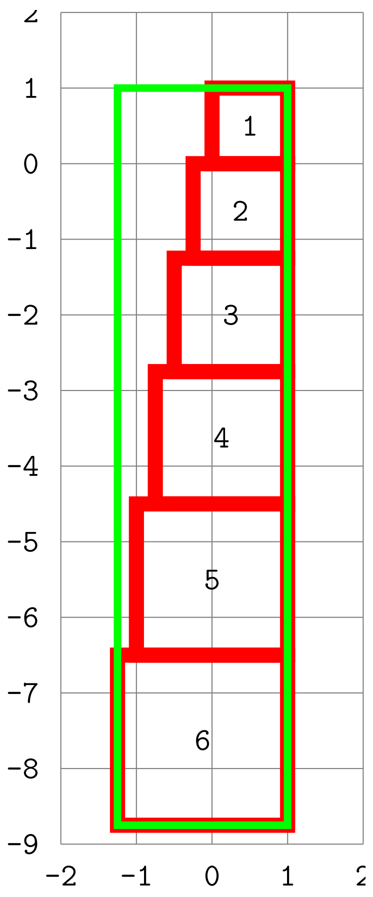
tmp/eb7c36a72aa7aeb861f538cf2fb3873a.pdf top-to-bottom, align left:

```
from latextool_basic import *
p = Plot()

c = RectContainer(x=1, y=1, align='left', direction='top-to-bottom')
for i, x in enumerate([1,2,3,4,5,6]):
    c += Rect2(x0=0, y0=0, x1=1+i/4.0, y1=1+i/4.0, linecolor='red', linewid
th=0.2,
               label=r'{\texttt{%s}}' % x)
p += c

# boundary of container
p += Rect2(x0=c.x0, y0=c.y0, x1=c.x1, y1=c.y1, linewidth=0.1, linecolor='gr
een')

p += Grid()
print(p)
```



tmp/2d34561aacc7e24c12eb38c0e1a86e63.pdf top-to-bottom, align right:

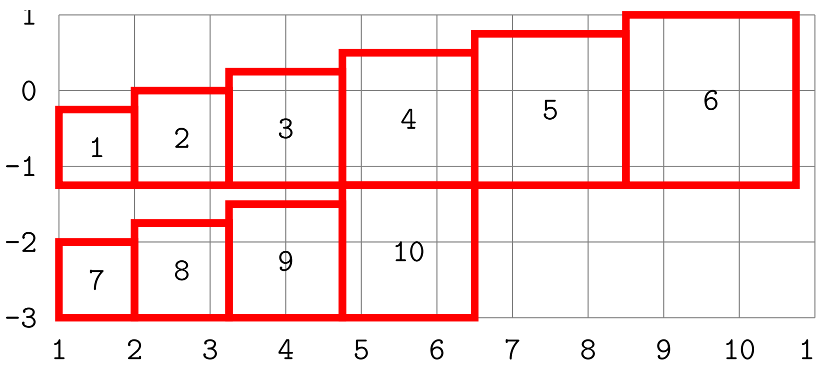
```
from latextool_basic import *
p = Plot()

c = RectContainer(x=1, y=1, align='right', direction='top-to-bottom')
for i, x in enumerate([1,2,3,4,5,6]):
    c += Rect2(x0=0, y0=0, x1=1+i/4.0, y1=1+i/4.0, linecolor='red', linewid
th=0.2,
               label=r'{\texttt{%s}}' % x)
p += c

# boundary of container
p += Rect2(x0=c.x0, y0=c.y0, x1=c.x1, y1=c.y1, linewidth=0.1, linecolor='gr
een')


p += Grid()
print(p)
```



tmp/13dfef2a97de09e644bcad0509714efa.pdf

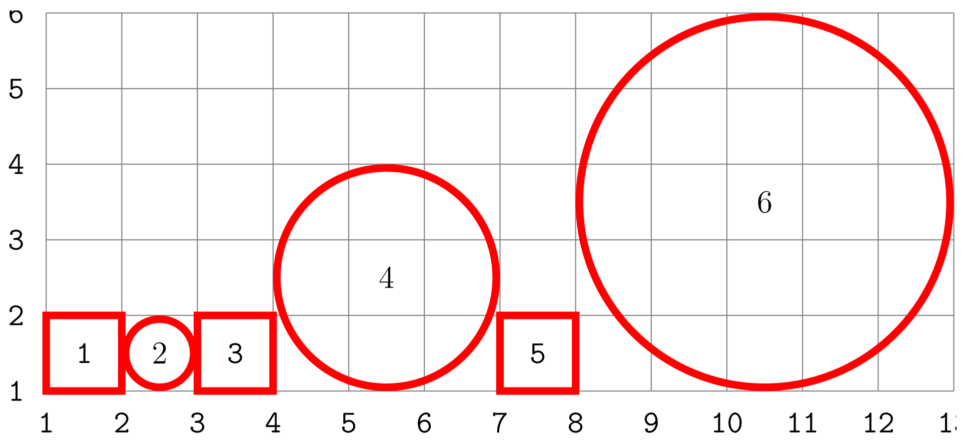
2-d stack

```
from latextool_basic import *
p = Plot()

c0 = RectContainer(x=1, y=1, align='bottom', direction='left-to-right')
for i, x in enumerate([1,2,3,4,5,6]):
    c0 += Rect2(x0=0, y0=0, x1=1+i/4.0, y1=1+i/4.0, linecolor='red', linewi
dth=0.1,
               label=r'{\texttt{%s}}' % x)

c1 = RectContainer(x=1, y=1, align='bottom', direction='left-to-right')
for i, x in enumerate([7,8,9,10]):
    c1 += Rect2(x0=0, y0=0, x1=1+i/4.0, y1=1+i/4.0, linecolor='red', linewi
dth=0.1,
               label=r'{\texttt{%s}}' % x)

C = RectContainer(x=1, y=1, align='left', direction='top-to-bottom')
C += c0; c0.x = c0.x0; c0.y = c0.y0; c0.layout()
C += c1; c1.x = c1.x0; c1.y = c1.y0; c1.layout()

p += C;
p += Grid()
print(p)
```



tmp/7a07ae3f99f9db0a91132bf36a69be62.pdf

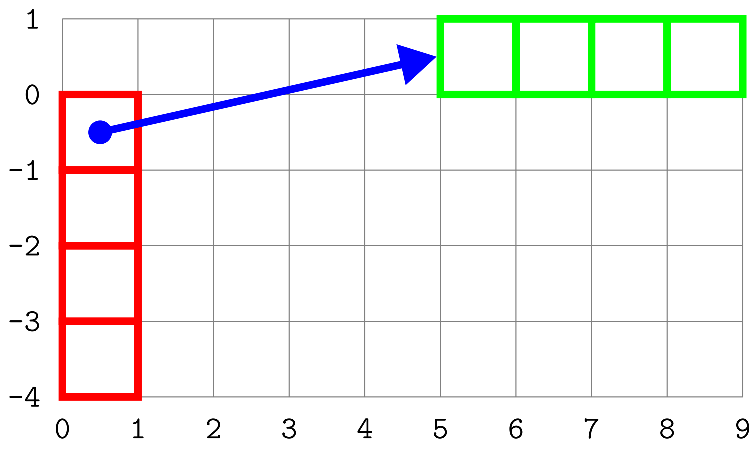
```
from latextool_basic import *
p = Plot()

c = RectContainer(x=1, y=1)
for i, x in enumerate([1,2,3,4,5,6]):
    if i % 2 == 0:
        c += Rect2(x0=0, y0=0, x1=1, y1=1, linecolor='red', linewidth=0.1,
                   label=r'{\texttt{%s}}' % x)
    else:
        c += Circle(x=0.5, y=0.5, r=0.5 + i/2, label=x, linecolor='red', li
newidth=0.1)

p += c
p += Grid()
print(p)
```



tmp/70352fdaf4f31a18b51eb36cf58500fa.pdf

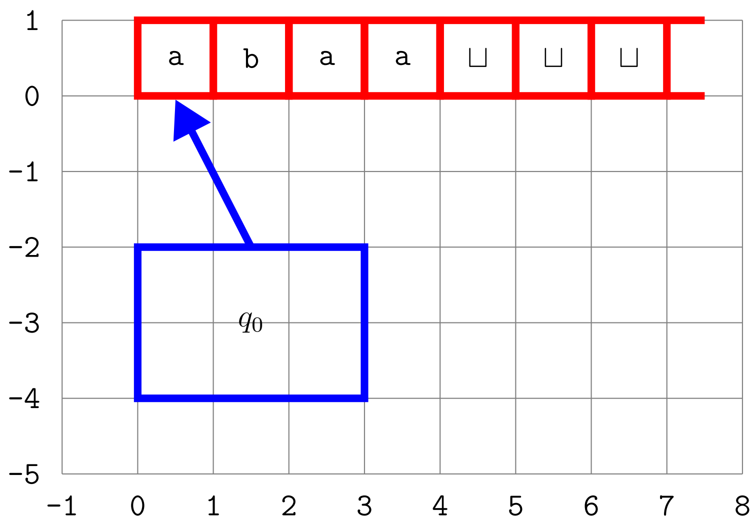
Vertical array and pointers

```
L = 0.1 # line width
from latextool_basic import *
p = Plot()


v = RectContainer(x=0, y=0, align='left', direction='top-to-bottom')
for i in ['','','','']:
    v += Rect2(0,0,1,1,s=i, linewidth=0.1, linecolor='red')

p += v

h = RectContainer(x=5, y=0, align='bottom', direction='left-to-right')
for i in ['','','','']:
    h += Rect2(0,0,1,1,s=i, linewidth=0.1, linecolor='green')

p += h

p += Line(x0=v[0].centerx(), y0=v[0].centery(),
          x1=h[0].leftx(), y1=h[0].centery(),
          linewidth=0.1, linecolor='blue',
          endstyle='->', startstyle='dot', r=0.01)

p += Grid()
print(p)
```



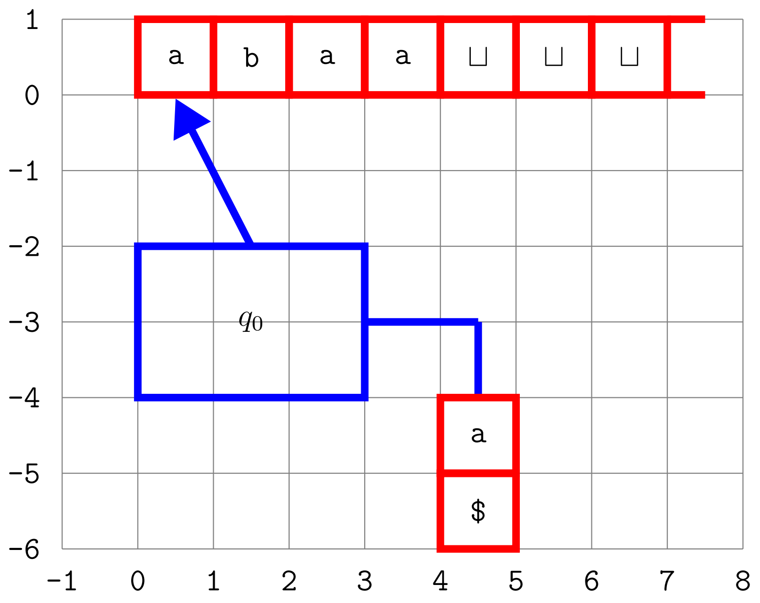
DFA/NFA:

```
from latextool_basic import *
p = Plot()

h = RectContainer(x=0, y=0, align='bottom', direction='left-to-right')
for i in ['a','b','a','a','$\sqcup$','$\sqcup$','$\sqcup$',]:
    h += Rect2(0,0,1,1,
               label= r'\texttt{%s}' % i,
               linewidth=0.1, linecolor='red')
p += h

p += Line(x0=h[-1].x1, y0=h[-1].y1, x1=h[-1].x1+0.5, y1=h[-1].y1,
          linewidth=0.1, linecolor='red')

p += Line(x0=h[-1].x1, y0=h[-1].y0, x1=h[-1].x1+0.5, y1=h[-1].y0,
          linewidth=0.1, linecolor='red')

dfa = Rect2(0,-4,3,-2, linewidth=0.1, linecolor='blue', label='$q_0$')
p += dfa

p += Line(x0=dfa.centerx(), y0=dfa.y1,
          x1=h[0].centerx(), y1=h[0].bottomy(),
          linewidth=0.1, linecolor='blue', endstyle='->')

p += Grid()
print(p)
```



PDA:

```
from latextool_basic import *
p = Plot()
h = RectContainer(x=0, y=0, align='bottom', direction='left-to-right')
for i in ['a','b','a','a','$\sqcup$','$\sqcup$','$\sqcup$',]:
    h += Rect2(0,0,1,1,
               label= r'\texttt{%s}' % i,
               linewidth=0.1, linecolor='red')
p += h

p += Line(x0=h[-1].x1, y0=h[-1].y1, x1=h[-1].x1+0.5, y1=h[-1].y1,
          linewidth=0.1, linecolor='red')
p += Line(x0=h[-1].x1, y0=h[-1].y0, x1=h[-1].x1+0.5, y1=h[-1].y0,
          linewidth=0.1, linecolor='red')

pda = Rect2(0,-4,3,-2, linewidth=0.1, linecolor='blue', label='$q_0$')
p += pda

p += Line(x0=pda.centerx(), y0=pda.y1,
          x1=h[0].centerx(), y1=h[0].bottomy(),
          linewidth=0.1, linecolor='blue', endstyle='->')

v = RectContainer(x=4, y=-4, align='left', direction='top-to-bottom')
for i in ['a', '\$',]:
    v += Rect2(0,0,1,1,
           label= r'\texttt{%s}' % i, linewidth=0.1, linecolor='red')
p += v

p += Line(x0=pda.x1,y0=pda.centery(), x1=v[0].centerx(), y1=pda.centery(),
          linewidth=0.1, linecolor='blue')
p += Line(x0=v[0].centerx(),y0=pda.centery(),x1=v[0].centerx(),y1=v[0].topy
(),
          linewidth=0.1, linecolor='blue')

p += Grid()
print(p)
```

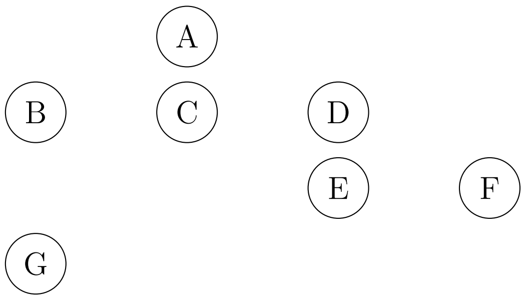

1
0
-1
-2
-3
-4
-5
-6
-1
0
1
2
3
4
5
6
7
8
a
b
a
a
␣
␣
␣
q_0
a
$

debug: positi

30 positions

The `positions` function computes positions.

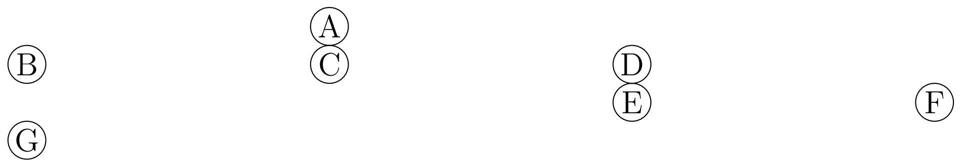
```
from latextool_basic import positions
d = positions(r'''
  A
B C D
    E F
G
''')

from latextool_basic import Plot, Circle
p = Plot()
for label, pos in d.items():
    x, y = pos
    p += Circle(x=x, y=y, r=0.4, label=label)
print(p)
```



tmp/07c19dd4cef2ae30b3195b0606a336b3.pdf

You can scale the x and y axes:

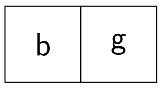
```
from latextool_basic import positions
d = positions(r'''
  A
B C D
    E F
G
''', xscale=2.0, yscale=0.5)

from latextool_basic import Plot, circle
p = Plot()
for label, pos in d.items():
    x, y = pos
    p.add(circle, x=x, y=y, r=0.25, label=label)
print(p)
```



tmp/b7c17c07a4629b0fe6760a1021dfe9a4.pdf debug: vphan

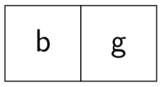
31 vphantom

no vphantom:



```
from latextool_basic import *
p = Plot()
p += Rect(x0=0, y0=0, x1=1, y1=1,
          label=r'{\textsf{%s}}' % "b")
p += Rect(x0=1, y0=0, x1=2, y1=1,
          label=r'{\textsf{%s}}' % "g")
print(p)
```

with vphantom:



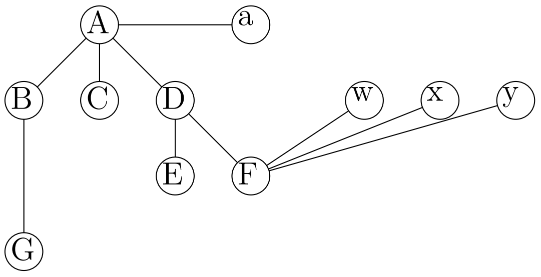
```
from latextool_basic import *
p = Plot()
p += Rect(x0=0, y0=0, x1=1, y1=1,
          label=r'{\vphantom{bg}\textsf{%s}}' % "b")
p += Rect(x0=1, y0=0, x1=2, y1=1,
          label=r'{\vphantom{bg}\textsf{%s}}' % "g")
print(p)
```

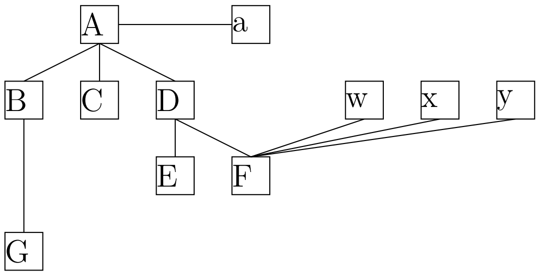
debug: chap-

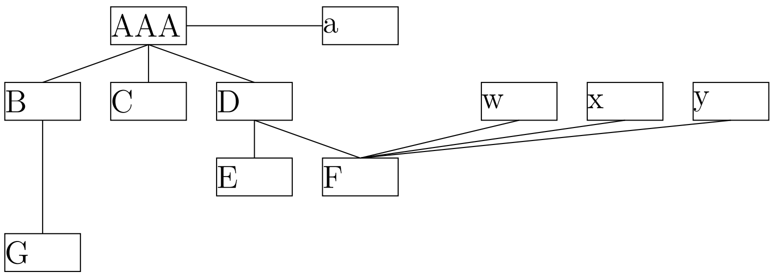
32 Trees

debug: tree.te

32.1 tree



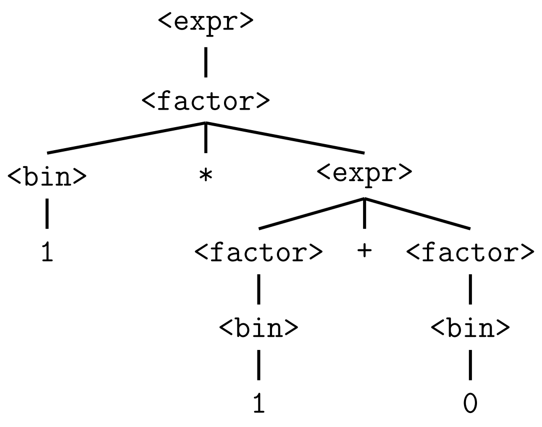


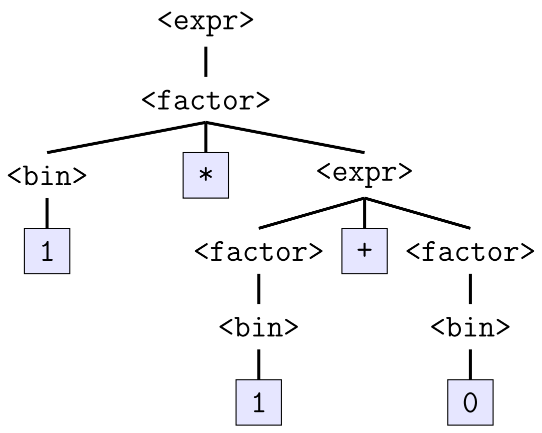


Auto adjust node position

32.2 Parse tree example

```
<expr> => <factor>
       => <bin> * <expr>
       => 1 * <expr>
       => 1 * <factor> + <factor>
       => 1 * <bin> + <factor>
       => 1 * 1 + <factor>
       => 1 * 1 + <bin>
       => 1 * 1 + 0
```

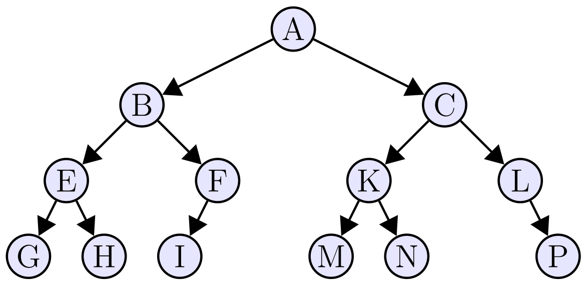




debug: tree2.

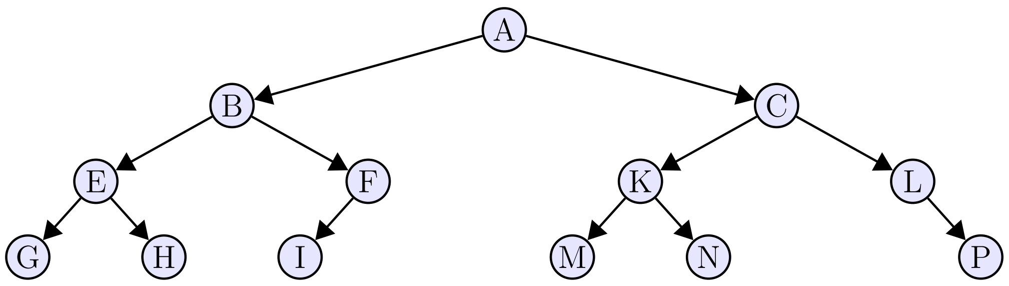
32.3 Binary tree

```
from latextool_basic import *

p = Plot()
edges={'A':['B','C'],
       'B':['E','F'],
       'E':['G','H'],
       'F':['I'],
       'C':['K','L'],
       'K':['M','N'],
       'L':['','P']}

pos = bintreepositions(edges=edges)

Graph.r = 0.3
for k,(x,y) in pos.items():
    p += Graph.node(x=x, y=y, name=k, label=k)

for k,v in edges.items():
    for x in v:
        if x in [None,'']: continue
        p += Graph.arc(names=[k,x])

print(p)
```



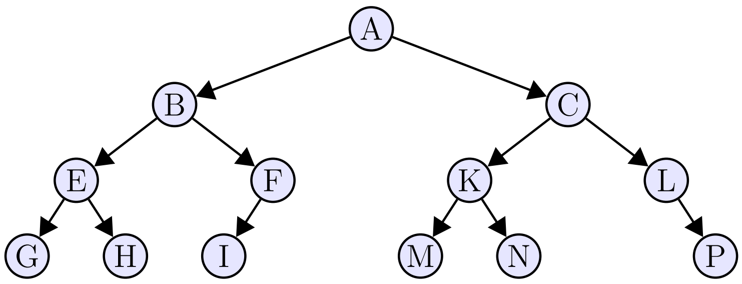
Changing separation between nodes:

```
from latextool_basic import *

p = Plot()
edges={'A':['B','C'],
       'B':['E','F'],
       'E':['G','H'],
       'F':['I'],
       'C':['K','L'],
       'K':['M','N'],
       'L':['','P']}

pos = bintreepositions(edges=edges, node_hsep=1, node_vsep=1)

Graph.r = 0.3
for k,(x,y) in pos.items():
    p += Graph.node(x=x, y=y, name=k, label=k)

for k,v in edges.items():
    for x in v:
        if x in [None,'']: continue
        p += Graph.arc(names=[k,x])

print(p)
```



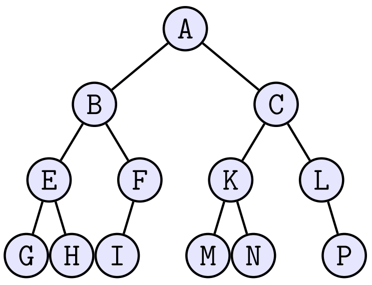
Changing separation between nodes:

```
from latextool_basic import *

p = Plot()
edges={'A':['B','C'],
       'B':['E','F'],
       'E':['G','H'],
       'F':['I'],
       'C':['K','L'],
       'K':['M','N'],
       'L':['','P']}

pos = bintreepositions(edges=edges, node_hsep=0.5, node_vsep=1)

Graph.r = 0.3
for k,(x,y) in pos.items():
    p += Graph.node(x=x, y=y, name=k, label=k)

for k,v in edges.items():
    for x in v:
        if x in [None,'']: continue
        p += Graph.arc(names=[k,x])

print(p)
```



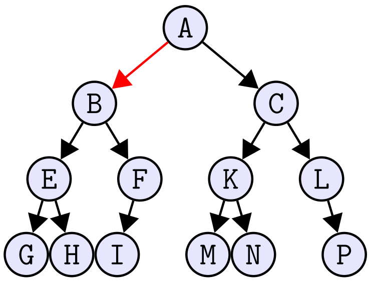
32.4 bintree function

```
from latextool_basic import *

p = Plot()
edges={'A':['B','C'],
       'B':['E','F'],
       'E':['G','H'],
       'F':['I'],
       'C':['K','L'],
       'K':['M','N'],
       'L':['','P']}

bintree(p, edges=edges)

print(p)
```



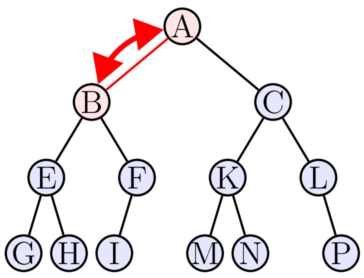
bintree function with edge function

```
from latextool_basic import *

p = Plot()
edges={'A':['B','C'],
       'B':['E','F'],
       'E':['G','H'],
       'F':['I'],
       'C':['K','L'],
       'K':['M','N'],
       'L':['','P']}

def edge(names):
    if names==['A','B']:
        return Graph.arc(names=names, linecolor='red')
    else:
        return Graph.arc(names=names)
bintree(p, edges=edges, edge=edge)

print(p)
```



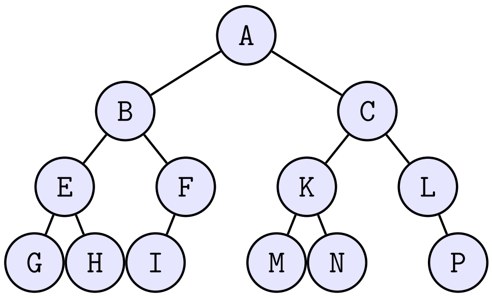
bintree function with node function

```
from latextool_basic import *

p = Plot()
edges={'A':['B','C'],
       'B':['E','F'],
       'E':['G','H'],
       'F':['I'],
       'C':['K','L'],
       'K':['M','N'],
       'L':['','P']}

def node(x, y, name, l):
    if name in ['A', 'B']:
        return Graph.node(x=x, y=y, r=0.25, name=name, label=l, background=
'red!10')
    else:
        return Graph.node(x=x, y=y, r=0.25, name=name, label=l)
def edge(names):
    if names==['A','B']:
        return Graph.edge(names=names, linecolor='red')
    else:
        return Graph.edge(names=names)

bintree(p, edges=edges, edge=edge, node=node)

p += Graph.edge(names=['A', 'B'],
     bend_right=30,
     endstyle='>', startstyle='>', linecolor='red', linewidth=0.07)

print(p)
```



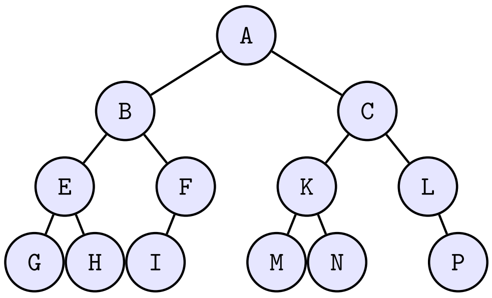
32.5 BinTree class

```
from latextool_basic import *

p = Plot()
edges={'A':['B','C'],
       'B':['E','F'],
       'E':['G','H'],
       'F':['I'],
       'C':['K','L'],
       'K':['M','N'],
       'L':['','P']}

BinTree.run(p, edges=edges)

print(p)
```



Using the BinTree class with label function

```
from latextool_basic import *

p = Plot()
edges={'A':['B','C'],
       'B':['E','F'],
       'E':['G','H'],
       'F':['I'],
       'C':['K','L'],
       'K':['M','N'],
       'L':['','P']}
def label(x): return r'{\footnotesize \texttt{%s}}' % x
BinTree.run(p, edges=edges, label=label)

print(p)
```



debug: chap-

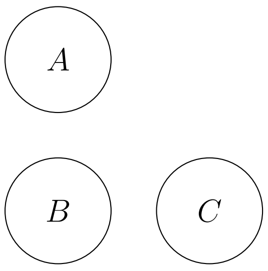
33 Graphs

debug: graph

34 Graphs

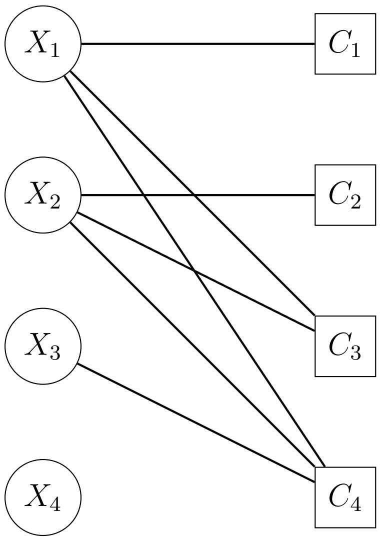
34.1 The `graph` function

```
\begin{python}
from latextool_basic import graph
print(graph(layout="""
A
B C
"""))
\end{python}
```

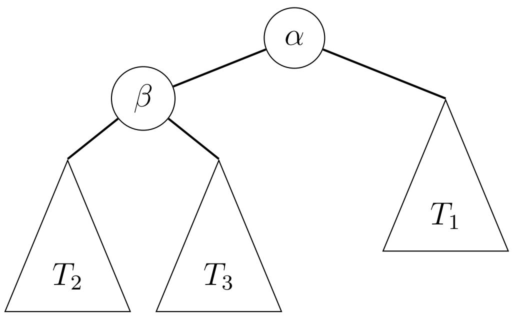


34.2 The `graph` function:nodes and labels

```
\begin{python}
from latextool_basic import graph
print(graph(layout='''
     X1  C1
     X2  C2
     X3  C3
     X4  C4
''',
minimum_size='8mm',
edges='X1-C1,X2-C2,X1-C3,X2-C3,X1-C4,X2-C4,X3dashedC4',
X1='label=$X_1$',
X2='label=$X_2$',
X3='label=$X_3$',
X4='label=$X_4$',
C1='shape=rectangle, label=$C_1$',
C2='shape=rectangle, label=$C_2$',
C3='shape=rectangle, label=$C_3$',
C4='shape=rectangle, label=$C_4$',
))
\end{python}
```

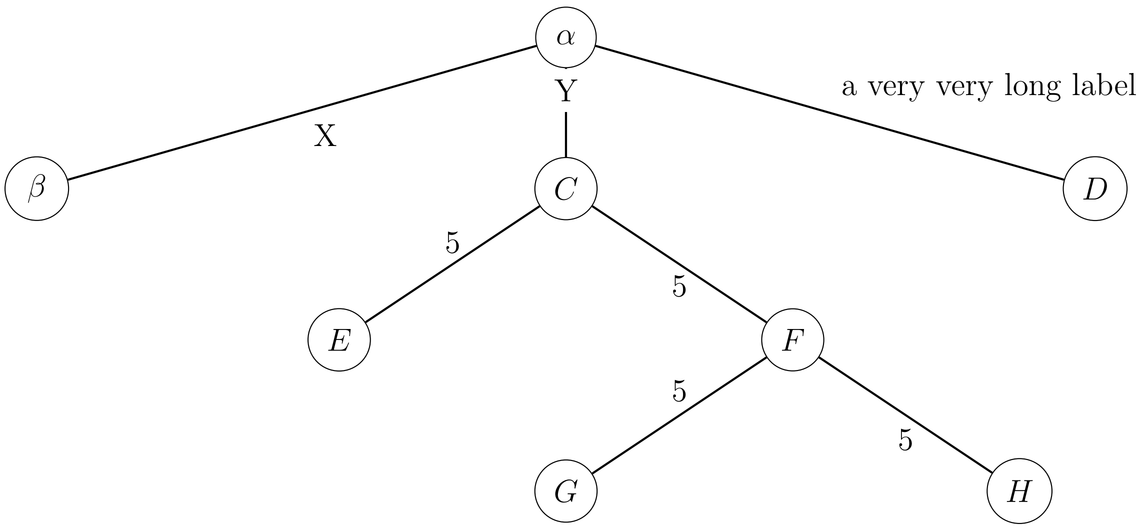


34.3 The `graph` function: isoceles triangle as subtree



34.4 The graph function: edge labels

```
from latextool_basic import graph
print(graph(layout='''
          A
   B      C      D
       E     F
          G     H
       I
''',
minimum_size='8mm',
edges='A-B,A-C,A-D,C-E,C-F,F-G,F-H,G-I',
A=r'label=$\alpha$',
B=r'label=$\beta$',
edge_label={('A','B'):{'label':'X'},
            ('A','C'):{'label':'Y', 'style':'pos=0.25,fill=white'},
            ('A','D'):{'label':'a very very long label'},
            ('C','E'):{'label':'5', 'style':'pos=0.5,above'},
            ('C','F'):{'label':'5', 'style':'pos=0.5,below'},
            ('F','G'):{'label':'5', 'style':'pos=0.5,above,inner sep=1mm,circle'},
            ('F','H'):{'label':'5', 'style':'pos=0.5,below,inner sep=1mm,circle'},
            ('G','I'):{'label':'abcdef', 'style':'sloped'},
           },
))
```

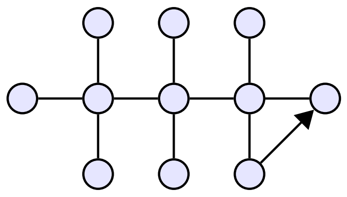


http://tex.stackexchange.com/questions/86386/edge-nodes-auto-and-node-distance

debug:
some-graphs.

34.5 Graph class

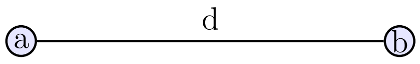
```
from latextool_basic import *
p = Plot()
for i in [1,2,3]:
    p += Graph.node(p, x=i, y=1, name='a%s' % i)
    p += Graph.node(p, x=i, y=-1, name='c%s' % i)

for i in range(5):
    p += Graph.node(p, x=i, y=0, name='b%s' % i)

p += Graph.edge(names=['b%s' % i for i in range(5)])
for i in [1,2,3]:
    p += Graph.edge(names=['a%s' % i, 'b%s' % i])
    p += Graph.edge(names=['c%s' % i, 'b%s' % i])

p += Graph.arc(names=['c3', 'b4'])

print(p)
```



34.6 Edge label anchor

```
from latextool_basic import *
p = Plot()
p += Graph.node(x=0, y=0, label='a', name='0')
p += Graph.node(x=5, y=0, label='b', name='1')
p += Graph.edge(names=['0','1'], label='d', anchor='above')

print(p)
```

```
from latextool_basic import *
p = Plot()
p += Graph.node(x=0, y=0, label='a', name='0')
p += Graph.node(x=5, y=0, label='b', name='1')
p += Graph.edge(names=['0','1'], label='d', anchor='above')

print(p)
```



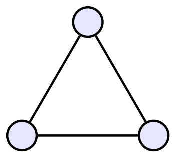
34.7 Cycle graphs

Odd Cycle.

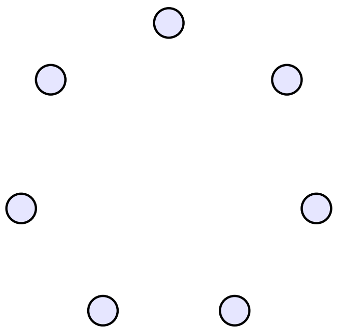
The node name is of the form `[radius]_[i]` where `i` is the index of node going in an anticlockwise direction, starting at degree `startdegree`.

By default, the first node is at `startdegree=90` degrees.

```
from latextool_basic import *
p = Plot()
cyclegraph(p, num=3, radius=1)
print(p)
```



```
from latextool_basic import *
p = Plot()
cyclegraph(p, num=7, radius=2, drawline=False)
print(p)
```

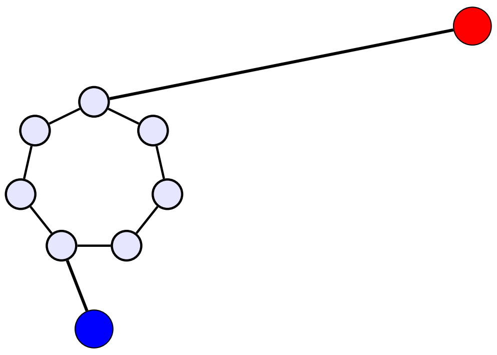


34.8 Name of nodes

Name of the nodes:

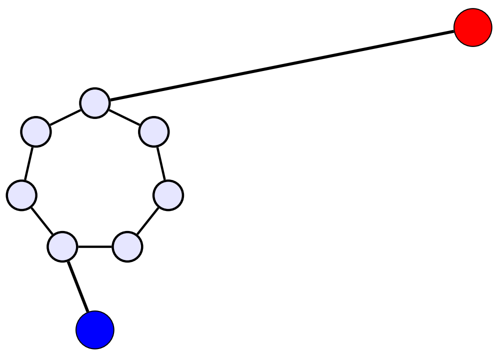
The nodes are names `'0'`, `'1'`, etc. in the anticlockwise direction where `'0'` is the node with angle `startdegree`. You can specify the `startdegree` in the `cyclegraph`. By default, when the number of nodes is even, the `startdegree` is 0 and when the number of nodes is odd, the `startdegree` is 90.

```
from latextool_basic import *
p = Plot()
cyclegraph(p, num=7, radius=1, drawline=True)
p += Circle(x=5, y=2, r=0.25, background='red', name='A')
p += Circle(x=0, y=-2, r=0.25, background='blue', name='B')
p += Line(names=['A', '0'])
p += Line(names=['B', '3'])
print(p)
```



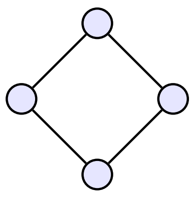
You can change the pgf/tikz name of the nodes by setting `names` dictionary in the `cyclegraph`. For instance if you want the first node to be named `'a'` instead of `0`, then `0:'a'` should be a key-value pair in `names`.

```
from latextool_basic import *
p = Plot()
cyclegraph(p, num=7, radius=1, drawline=True,
           names={0:'a',
                  3:'d',
                  })
p += Circle(x=5, y=2, r=0.25, background='red', name='A')
p += Circle(x=0, y=-2, r=0.25, background='blue', name='B')
p += Line(names=['A', 'a'])
p += Line(names=['B', 'd'])
print(p)
```

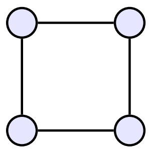


Even Cycle. By default, the starting degree is 0.

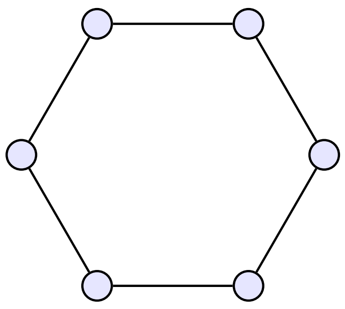
```
from latextool_basic import *
p = Plot()
cyclegraph(p, num=4, radius=1)
print(p)
```



```
from latextool_basic import *
p = Plot()
cyclegraph(p, num=4, radius=1, startdegree=45)
print(p)
```

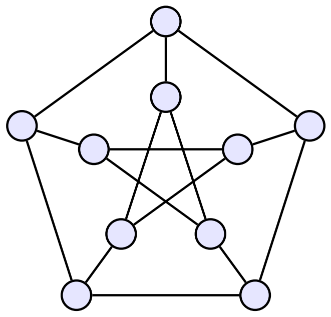


```
from latextool_basic import *
p = Plot()
cyclegraph(p, num=6, radius=2)
print(p)
```



34.9 Petersen graph

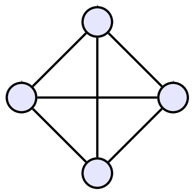
```
from latextool_basic import *
p = Plot()
petersen(p)
print(p)
```



34.10 Complete graphs

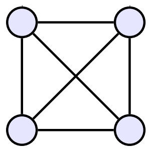
K_4:

```
from latextool_basic import *
p = Plot()
completegraph(p, num=4)
print(p)
```



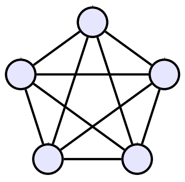
K_4:

```
from latextool_basic import *
p = Plot()
completegraph(p, num=4, startdegree=45)
print(p)
```



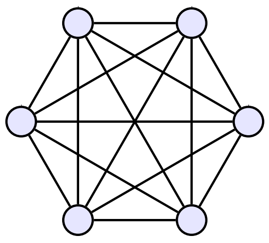
K_5:

```
from latextool_basic import *
p = Plot()
completegraph(p, num=5)
print(p)
```



K_6:

```
from latextool_basic import *
p = Plot()
completegraph(p, num=6, radius=1.5)
print(p)
```



34.11 Star graphs

The name of the center node is `'center'`.

```
from latextool_basic import *
p = Plot()
stargraph(p, num=5)
print(p)
```

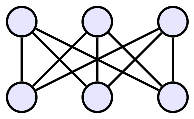
```
from latextool_basic import *
p = Plot()
stargraph(p, num=4)
print(p)
```

When there are two stars, you need to give the centers different names:

```
from latextool_basic import *
p = Plot()
stargraph(p, x=0, y=0, num=4, radius=1, names={'center':'c0'})
stargraph(p, x=2, y=0, num=4, radius=1, names={'center':'c1'})
print(p)
```

34.12 Bipartite graphs

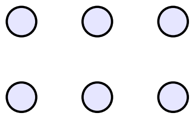
$K_{3,3}$:



```
from latextool_basic import *
p = Plot()
completebipartite(p=p, num1=3, num2=3)
print(p)
```

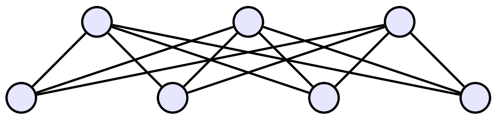
$K_{3,3}$ without edges:

```
from latextool_basic import *
p = Plot()
completebipartite(p=p, num1=3, num2=3, drawline=False)
print(p)
```



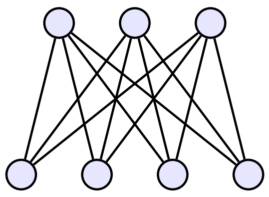
$K_{3,4}$:

```
from latextool_basic import *
p = Plot()
completebipartite(p=p, num1=3, num2=4, horsep=2)
print(p)
```



$K_{3,4}$:

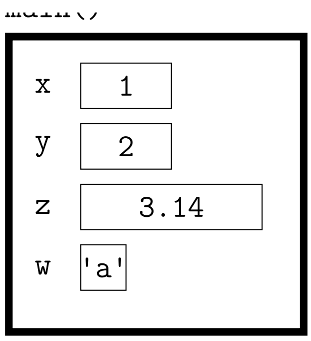
```
from latextool_basic import *
p = Plot()
completebipartite(p=p, num1=3, num2=4, versep=2)
print(p)
```



debug: frame

35 Frame

```
from latextool_basic import *
p = Plot()
env = [('x', 1), ('y', 2), ('z', 3.14), ('w', 'a')]
p.add(frame(env, top="main()"))
print(p)
```



tmp/676f2b6ad3cb833fe368eb4fd38f509d.pdf

debug: test-v
spacing.tex

36 Test Verbatim Spacing

First here's the vertical spacing between paragraphs. Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

TEST 1. Text, console environment, text:

```
hello world
```

Line after.

TEST 2. Text, console environment, console environment, text:

```
hello world
```

```
hello world
```

Line after. Probably won't be used. Probably too much spacing between the two environments.

TEST 3. Text, python environment with console function, text:

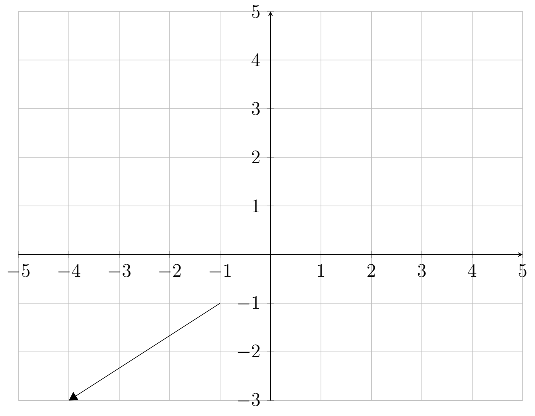
```
hello world
```

Line after.

debug: vec2d

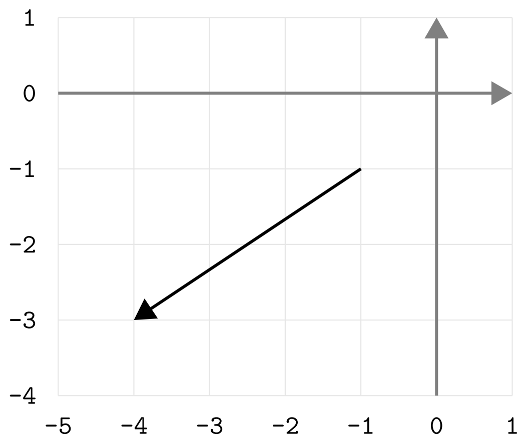
37 2D vector diagram

```
from latextool_basic import *
p, q = (-1,-1), (-4,-3)
print(plot(vector=[p,q]))
v = vec(p, q)
print(answer(answer=''))
```



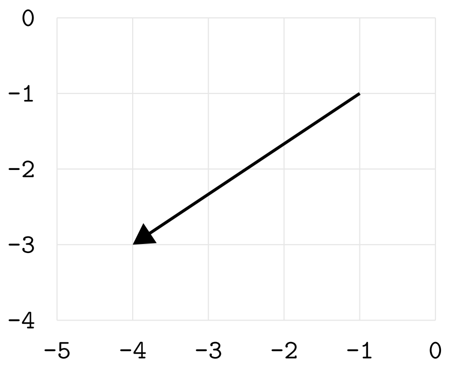
ANSWER: $\boxed{}$

```
from latextool_basic import *
plot = Plot()
p, q = (-1,-1), (-4,-3); vector = [p, q]; vectors = [vector]
vec2dplot(plot, vectors=[vector])
print(plot)
```



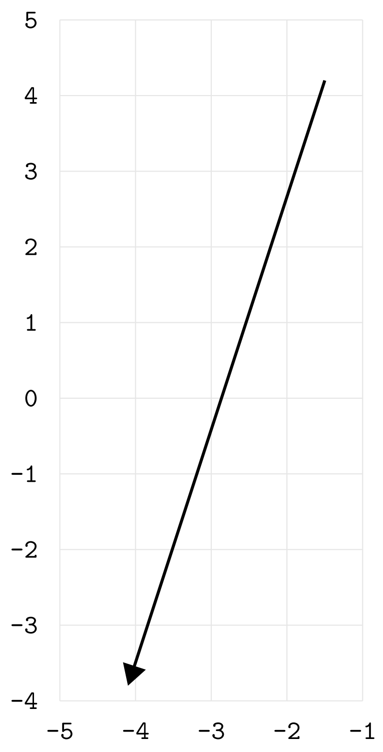
Without axes:

```
from latextool_basic import *
plot = Plot()
p, q = (-1,-1), (-4,-3); vector = [p, q]; vectors = [vector]
vec2dplot(plot, vectors=[vector], draw_axes=False)
print(plot)
```



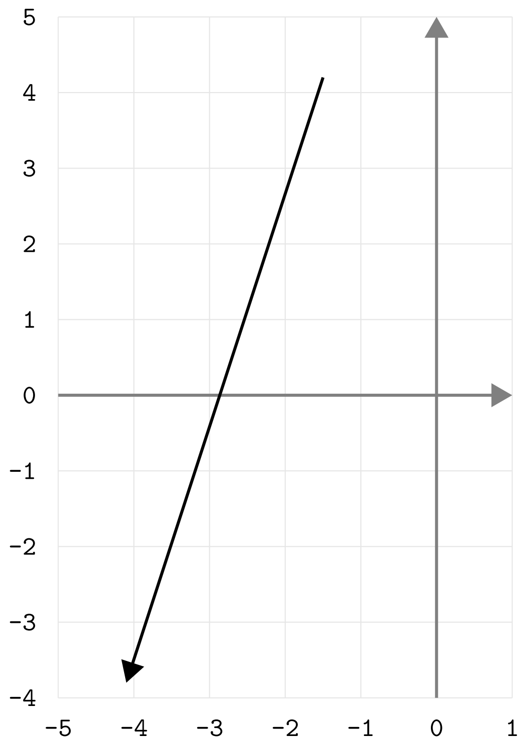
Without axes and vector with non integer endpoints:

```
from latextool_basic import *
plot = Plot()
p, q = (-1.5, 4.2), (-4.1, -3.8); vector = [p, q]; vectors = [vector]
vec2dplot(plot, vectors=[vector], draw_axes=False)
print(plot)
```



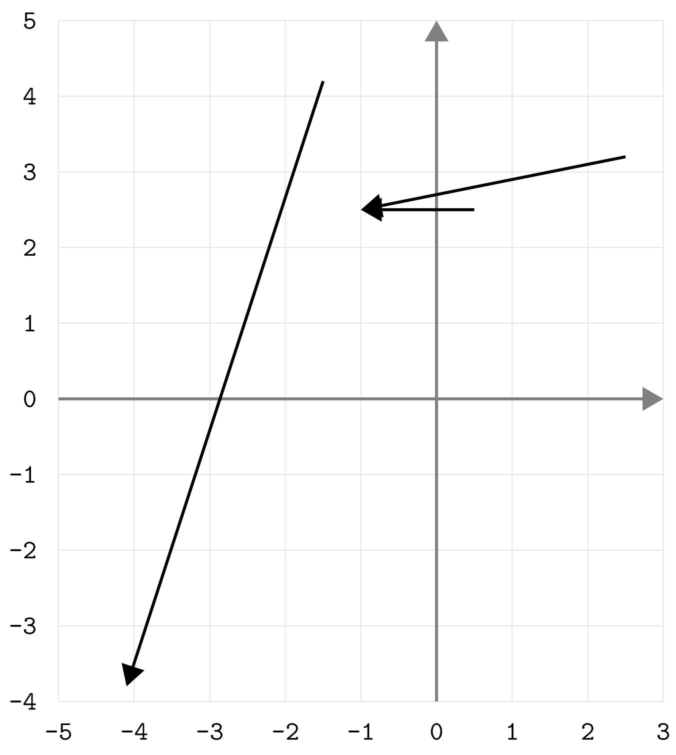
With axes and vector with non integer endpoints:

```
from latextool_basic import *
plot = Plot()
p, q = (-1.5, 4.2), (-4.1, -3.8); vector = [p, q]; vectors = [vector]
vec2dplot(plot, vectors=[vector])
print(plot)
```



With axes and multiple vectors with non integer endpoints:

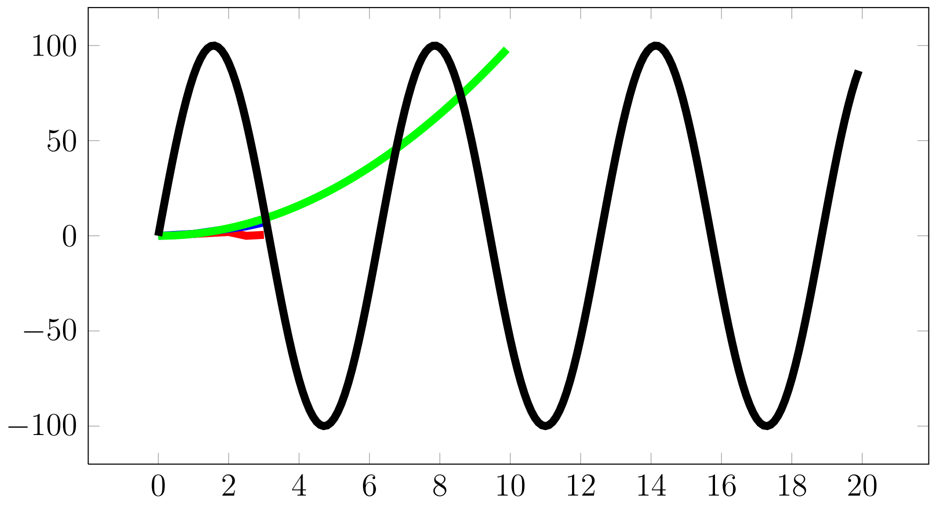
```
from latextool_basic import *
plot = Plot()
p, q = (-1.5, 4.2), (-4.1, -3.8); v0 = [p, q];
p, q = (2.5, 3.2), (-1, 2.5); v1 = [p, q];
p, q = (0.5, 2.5), (-1, 2.5); v2 = [p, q];
vectors = [v0, v1, v2]
vec2dplot(plot, vectors=vectors)
print(plot)
```



debug: line-g

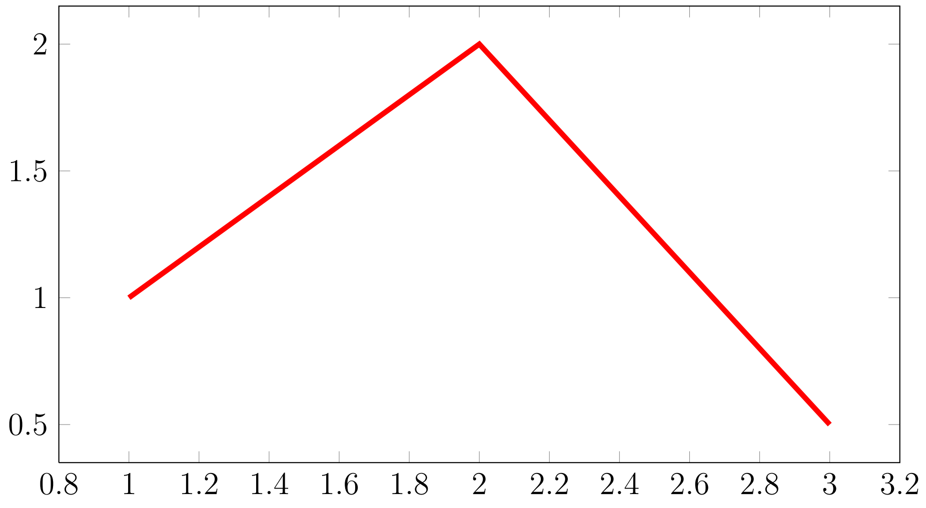
38 Line Graph

```
from math import sin
from latextool_basic import *
plot = FunctionPlot()

data = ((1, 1), (2, 2), (2.5, 0), (3, 0.5))
data2 = ((0, 0), (1, 1), (2.5, 5), (3, 7))
data3 = [(x/10.0, (x/10.0)**2) for x in range(0, 100)]
data4 = [(x/10.0, 100 * sin(x/10.0)) for x in range(0, 200)]
plot.add(data, line_width='3', color='red')
plot.add(data2, line_width='3', color='blue')
plot.add(data3, line_width='3', color='green')
plot.add(data4, line_width='3', color='black')
print(plot)
```



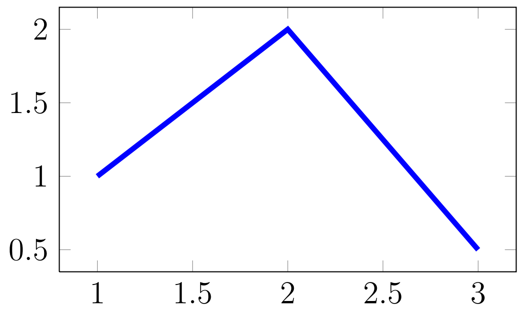
Line width and color:

```
from latextool_basic import *
plot = FunctionPlot()
plot.add(((1, 1), (2, 2), (3, 0.5)), line_width='2', color='red')
print(plot)
```



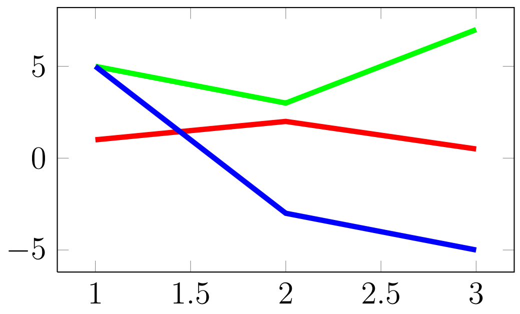
Width and height:

```
from latextool_basic import *
plot = FunctionPlot(width="3in", height="2in")
plot.add(((1, 1), (2, 2), (3, 0.5)), line_width='2', color='blue')
print(plot)
```



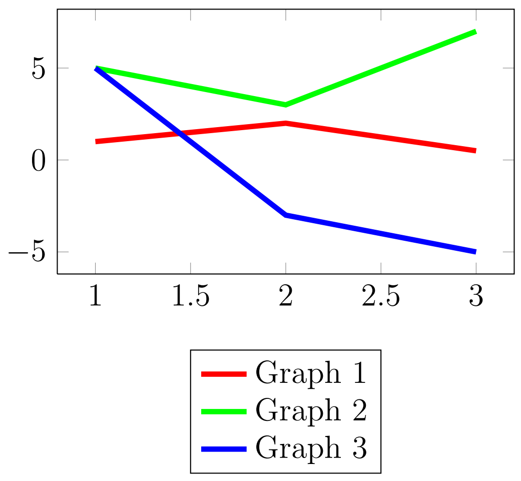
Multiple lines:

```
from latextool_basic import *
plot = FunctionPlot(width="3in", height="2in")
plot.add(((1, 1), (2, 2), (3, 0.5)), line_width='2', color='red')
plot.add(((1, 5), (2, 3), (3, 7)), line_width='2', color='green')
plot.add(((1, 5), (2, -3), (3, -5)), line_width='2', color='blue')
print(plot)
```



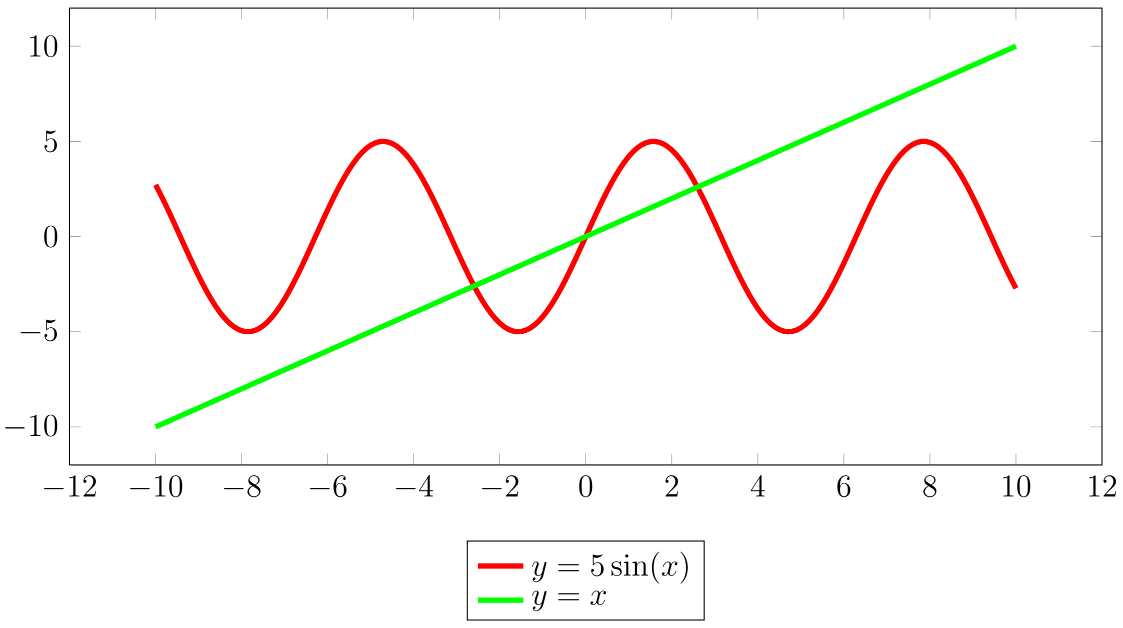
Legend:

```
from latextool_basic import *
plot = FunctionPlot(width="3in", height="2in")
plot.add(((1, 1), (2, 2), (3, 0.5)), line_width='2', color='red', legend='G
raph 1')
plot.add(((1, 5), (2, 3), (3, 7)), line_width='2', color='green', legend='G
raph 2')
plot.add(((1, 5), (2, -3), (3, -5)), line_width='2', color='blue', legend='
Graph 3')
print(plot)
```



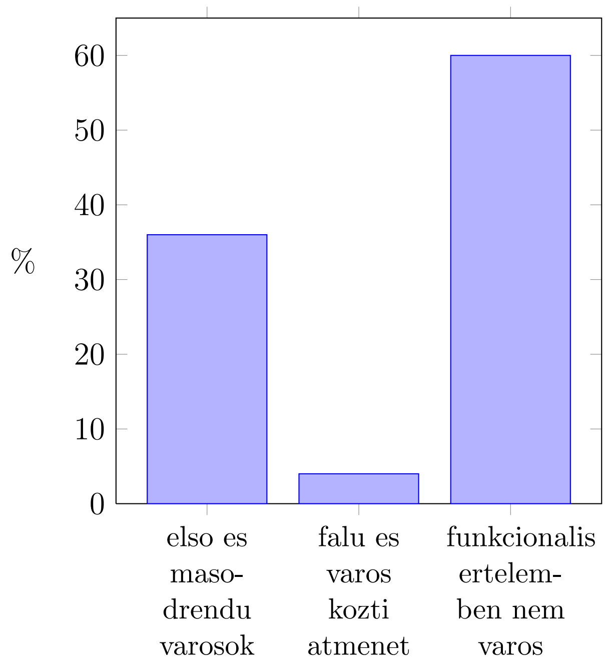
Legend:

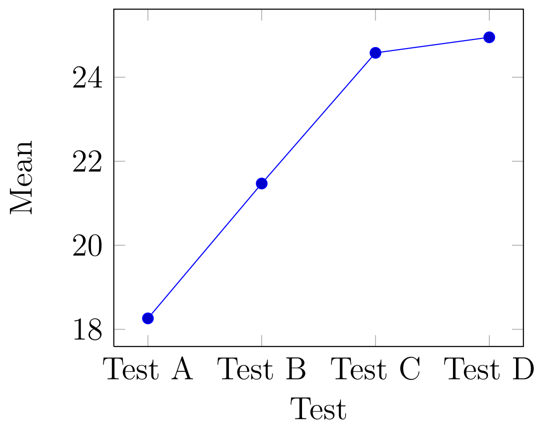
```
from math import sin
domain = [x / 10.0 for x in range(-100, 101)]
y_sinx = [(x, 5 * sin(x)) for x in domain]
y_x = [(x, x) for x in domain]

from latextool_basic import *
plot = FunctionPlot(width="6in", height="3in")
plot.add(y_sinx, line_width='2', color='red', legend=r'$y = 5 \sin(x)$')
plot.add(y_x, line_width='2', color='green', legend='$y = x$')
print(plot)
```

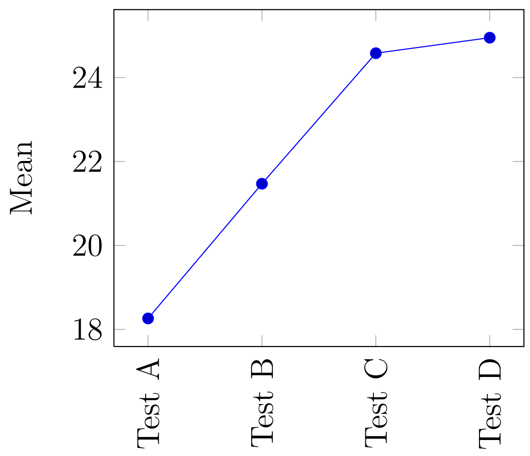


debug: line-g
nonnumeric.t

39 Bar chart, line chart

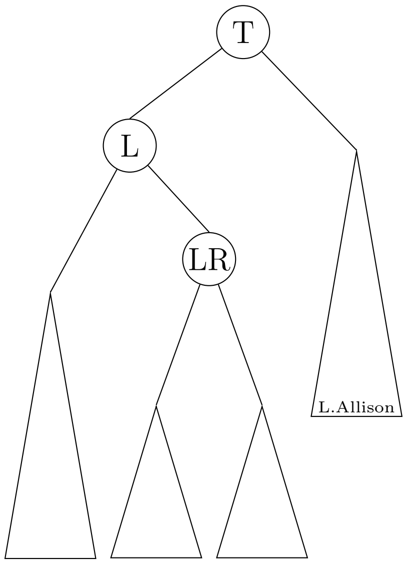





Mean
18
20
22
24
Test A
Test B
Test C
Test D

debug:
isosceles-trian

40 TEST ISOCELES TRIANGLE



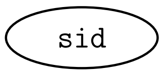
http://tex.stackexchange.com/questions/7862/triangle-node-with-adjustable-height

http://tex.stackexchange.com/questions/37462/placing-a-triangle-around-nodes-in-a-tre debug: er-dia

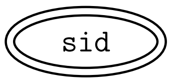
41 ER diagrams

41.1 Attribute

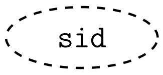
```
from latextool_basic import *; p = Plot()
p += ER.attrib(center=(0,0), name='sid', label=r'\texttt{sid}')
print(p)
```



```
from latextool_basic import *; p = Plot()
p += ER.attrib(center=(0,0), name='sid', label=r'\texttt{sid}',
               double=True)
print(p)
```

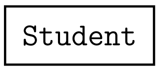


```
from latextool_basic import *; p = Plot()
p += ER.attrib(center=(0,0), name='sid', label=r'\texttt{sid}', linestyle='
dashed')
print(p)
```



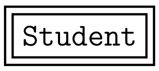
41.2 Entity

```
from latextool_basic import *; p = Plot()
p += ER.entity(center=(6,0), name='Student')
print(p)
```



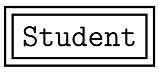
Entity with double lines:

```
from latextool_basic import *; p = Plot()
p += ER.entity(center=(6,0), name='Student', double=True)
print(p)
```



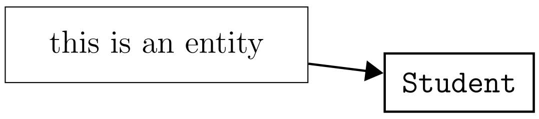
Entity with double lines:

```
from latextool_basic import *; p = Plot()
p += ER.entity(center=(6,0), name='Student', double=['Student'])
print(p)
```



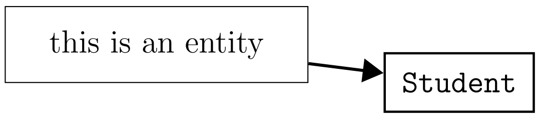
Entity name is a tikz node name:

```
from latextool_basic import *; p = Plot()
p += ER.entity(center=(6,0), name='Student')
p += Rect(0,0,4,1,name='r',label='this is an entity')
p += ER.line(names=['r', 'Student'], endstyle='>')
print(p)
```

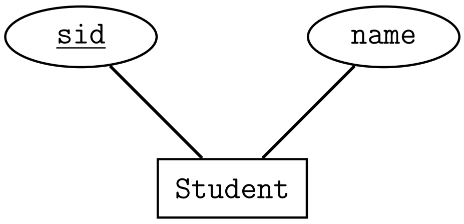


Entity name can be difference from label:

```
from latextool_basic import *; p = Plot()
p += ER.entity(center=(6,0), name='s', label='Student')
p += Rect(0,0,4,1,name='r',label='this is an entity')
p += Line(names=['r', 's'], endstyle='>')
print(p)
```

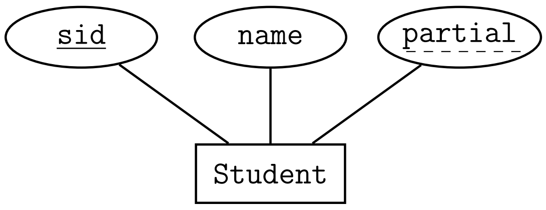

this is an entity
Student

```
from latextool_basic import *
p = Plot()

p += ER.attrib(center=(4,2), name='sid', label=r'\texttt{\underline{sid}}')
p += ER.attrib(center=(8,2), name='name', label=r'\texttt{name}')
p += ER.entity(center=(6,0), name='Student', label=r'\texttt{Student}')

p += Line(names=['sid', 'Student'])
p += Line(names=['name', 'Student'])
print(p)
```

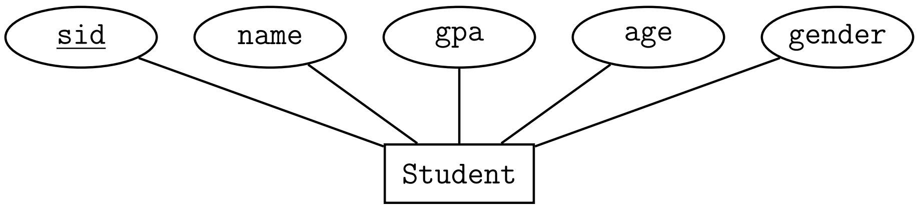


Entity with attributes (primary key and partial key):

```
from latextool_basic import *; p = Plot()
p += ER.entity(center=(0,0),
          name='Student',
          attribs=['sid', 'name', 'partial'], keys=['sid'], dasheds=['parti
al'])
print(p)
```

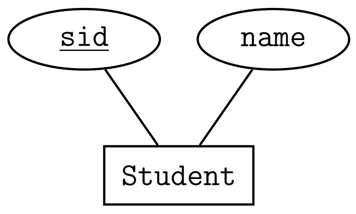


```
from latextool_basic import *; p = Plot()
p += ER.entity(center=(0,0),
          name='Student',
          attribs=['sid', 'name', 'gpa', 'age', 'gender'],
          keys=['sid'])
print(p)
```

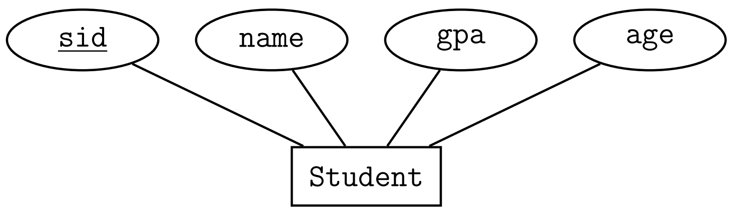


Entity with attribs on north (even number):

```
from latextool_basic import *; p = Plot()
p += ER.entity(center=(0,0),
          name='Student',
          attribs=['sid', 'name'], keys=['sid'])
print(p)
```

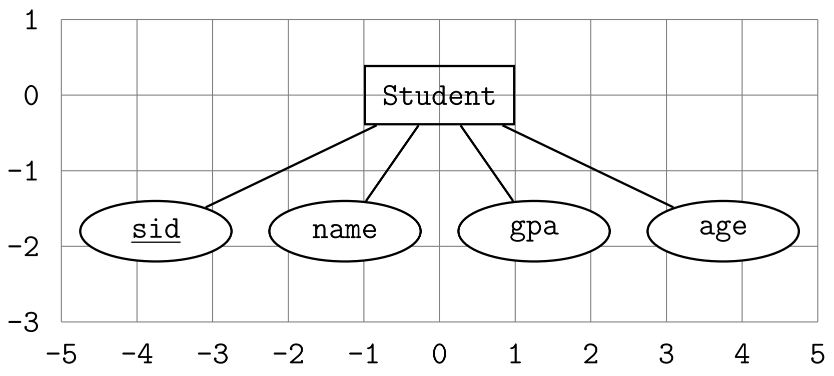


```
from latextool_basic import *; p = Plot()
p += ER.entity(center=(0,0),
          name='Student',
          attribs=['sid', 'name', 'gpa', 'age'], keys=['sid'])
print(p)
```



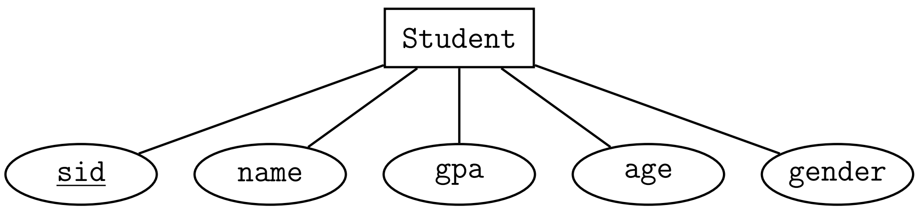
Even number of attributes on south:

```
from latextool_basic import *
p = Plot()
p += ER.entity(center=(0,0),
          name='Student',
          attribs=['sid', 'name', 'gpa', 'age'],
          keys=['sid'],
          anchor={'sid':'south', 'name':'south','gpa':'south','age':'south'
})
p += Grid(-5,-3,5,1)
print(p)
```

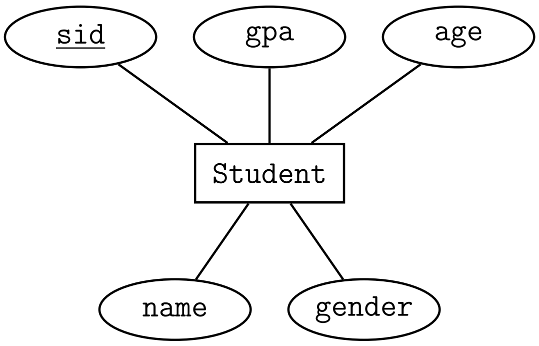


Odd number of attributes on south:

```
from latextool_basic import *
p = Plot()
p += ER.entity(center=(0,0),
          name='Student',
          attribs=['sid', 'name', 'gpa', 'age', 'gender'],
          keys=['sid'],
          anchor={'sid':'south', 'name':'south','gpa':'south','age':'south'
,'gender':'south'})
print(p)
```

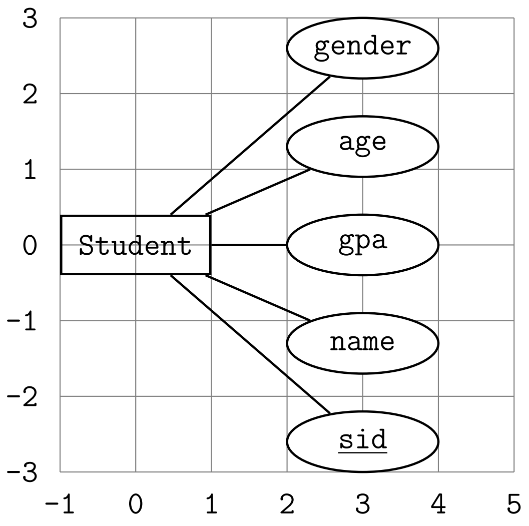


```
from latextool_basic import *
p = Plot()
p += ER.entity(center=(0,0),
          name='Student',
          attribs=['sid', 'name', 'gpa', 'age', 'gender'],
          keys=['sid'],
          anchor={'name':'south', 'gender':'south'})
print(p)
```



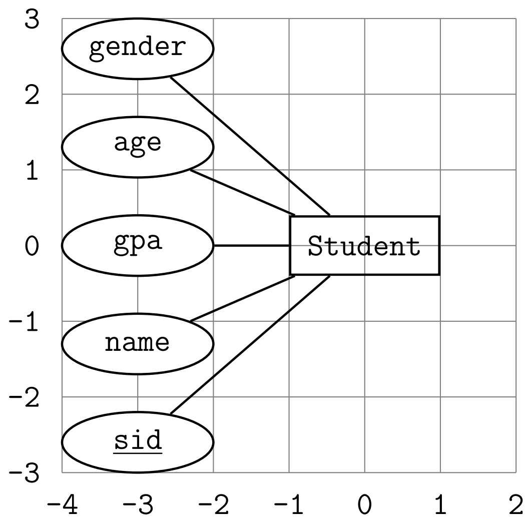
Attributes on east:

```
from latextool_basic import *
p = Plot()
p += ER.entity(center=(0,0),
          name='Student',
          attribs=['sid', 'name', 'gpa', 'age', 'gender'],
          keys=['sid'],
          anchor={'sid':'east', 'name':'east', 'gpa':'east',
                  'age':'east',
                  'gender':'east'})
p += Grid(-1, -3, 5, 3)
print(p)
```



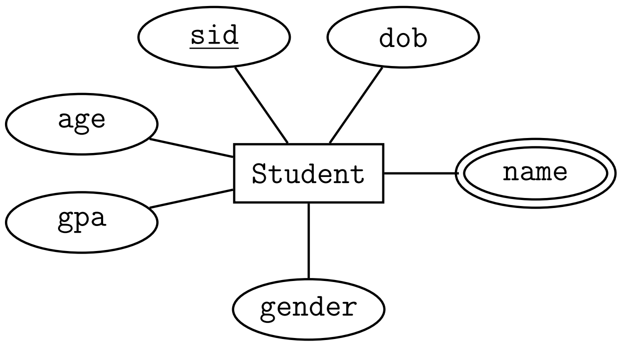
Attributes on west:

```
from latextool_basic import *
p = Plot()
p += ER.entity(center=(0,0),
          name='Student',
          attribs=['sid', 'name', 'gpa', 'age', 'gender'],
          keys=['sid'],
          anchor={'sid':'west', 'name':'west', 'gpa':'west',
                  'age':'west', 'gender':'west'})
p += Grid(-4, -3, 2, 3)
print(p)
```



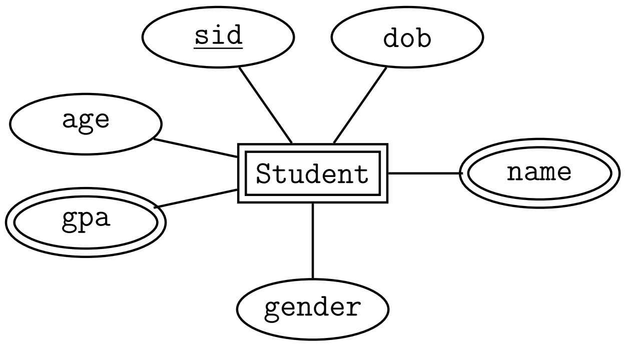
All directions:

```
from latextool_basic import *
p = Plot()
p += ER.entity(center=(0,0),
          name='Student',
          attribs=['sid', 'name', 'gpa', 'age', 'gender', 'dob'],
          keys=['sid'],
          double=['name'],
          anchor={'name':'east', 'gpa':'west',
                  'age':'west', 'gender':'south'})
print(p)
```



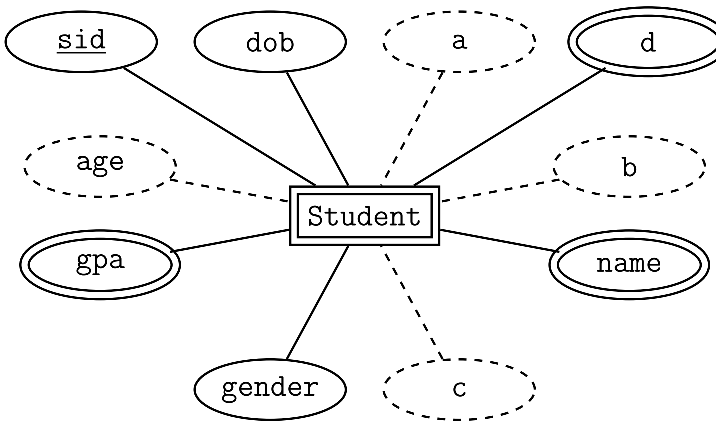
All directions and double:

```
from latextool_basic import *
p = Plot()
p += ER.entity(center=(0,0),
          name='Student',
          attribs=['sid', 'name', 'gpa', 'age', 'gender', 'dob'],
          keys=['sid'],
          double=['name', 'gpa', 'Student'],
          anchor={'name':'east', 'gpa':'west',
                  'age':'west', 'gender':'south'})
print(p)
```



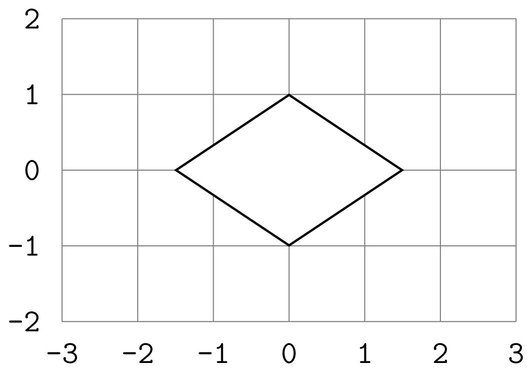
All directions and double and derived

```
from latextool_basic import *
p = Plot()
ER.attrib_dist = 1.5
p += ER.entity(center=(0,0),
          name='Student',
          attribs=['sid', 'name', 'gpa', 'age', 'gender', 'dob', 'a', 'b',
'c', 'd'],
          keys=['sid'],
          derived=['age', 'a', 'b', 'c'],
          double=['name', 'gpa', 'Student', 'd'],
          anchor={'name':'east', 'gpa':'west', 'b':'east', 'c':'south',
                  'age':'west', 'gender':'south'})
print(p)
```

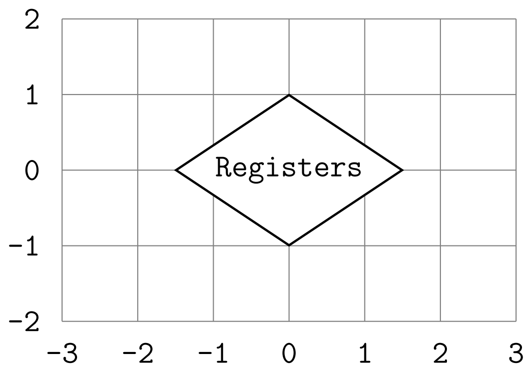


41.3 Relation

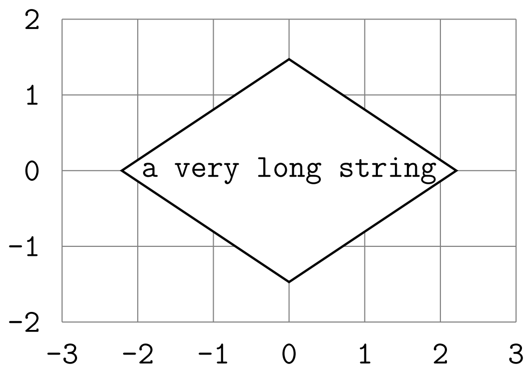
```
from latextool_basic import *
p = Plot()
p += ER.relation(center=(0,0), name='Registers', label='')
p += Grid(-3, -2, 3, 2)
print(p)
```



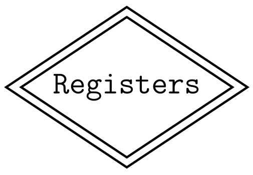
```
from latextool_basic import *
p = Plot()
p += ER.relation(center=(0,0), name='Registers', label='Registers')
p += Grid(-3, -2, 3, 2)
print(p)
```



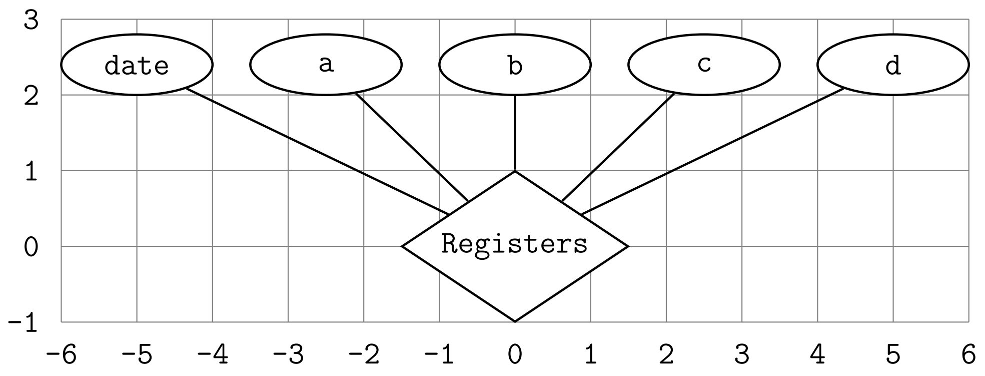
```
from latextool_basic import *
p = Plot()
p += ER.relation(center=(0,0), name='Registers', label='a very long string'
)
p += Grid(-3, -2, 3, 2)
print(p)
```



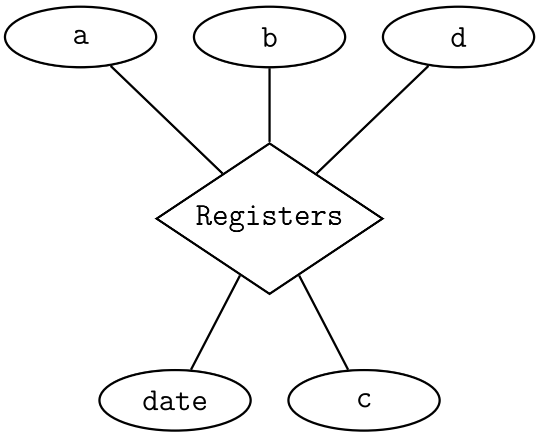
```
from latextool_basic import *
p = Plot()
p += ER.relation(center=(4,0), double=True,
                 name='Registers', label='Registers')
print(p)
```



```
from latextool_basic import *
p = Plot()
p += ER.relation(center=(0,0),
                 name='Registers',
                 attribs=['date','a','b','c','d'])
p += Grid(-6, -1, 6, 3)
print(p)
```

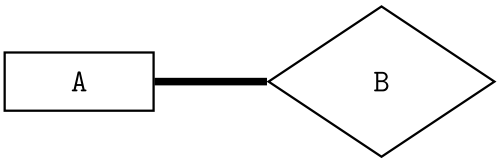


```
from latextool_basic import *
p = Plot()
p += ER.relation(center=(4,0),
                 name='Registers',
                 attribs=['date','a','b','c','d'],
                 anchor={'date':'south', 'c':'south'})
print(p)
```

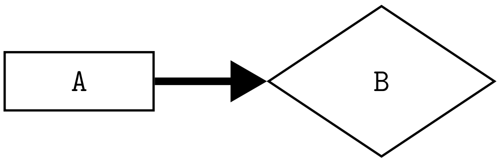


41.4 Entity and relation with different arrows

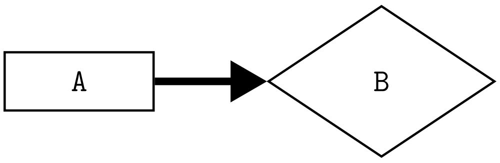
```
from latextool_basic import *
p = Plot()
p += ER.entity(center=(0,0), name='A')
p += ER.relation(center=(4,0), name='B')
p += ER.line(names=['A', 'B'],
             linewidth=ER.boldlinewidth)
print(p)
```



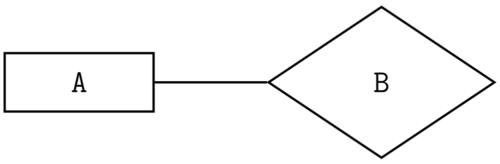
```
from latextool_basic import *
p = Plot()
p += ER.entity(center=(0,0), name='A')
p += ER.relation(center=(4,0), name='B')
p += ER.line(names=['A', 'B'],
             linewidth=ER.boldlinewidth, endstyle='>')
print(p)
```



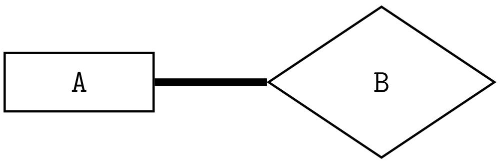
```
from latextool_basic import *
p = Plot()
p += ER.entity(center=(0,0), name='A')
p += ER.relation(center=(4,0), name='B')
p += ER.line(names=['A', 'B'],
             linewidth=ER.boldlinewidth, endstyle='>',
             arrowstyle='triangle')
print(p)
```



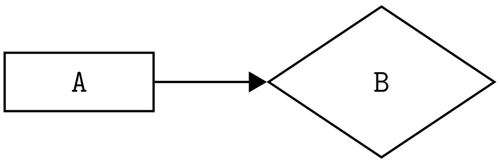
```
from latextool_basic import *
p = Plot()
p += ER.entity(center=(0,0), name='A')
p += ER.relation(center=(4,0), name='B')
p += ER.edge(names=['A', 'B'])
print(p)
```



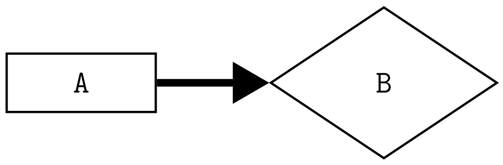
```
from latextool_basic import *
p = Plot()
p += ER.entity(center=(0,0), name='A')
p += ER.relation(center=(4,0), name='B')
p += ER.boldedge(names=['A', 'B'])
print(p)
```



```
from latextool_basic import *
p = Plot()
p += ER.entity(center=(0,0), name='A')
p += ER.relation(center=(4,0), name='B')
p += ER.arc(names=['A', 'B'])
print(p)
```

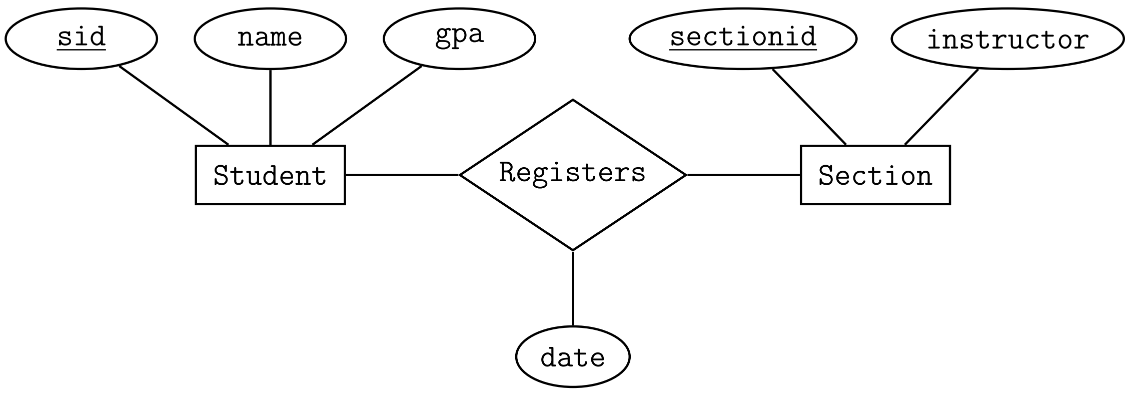


```
from latextool_basic import *
p = Plot()
p += ER.entity(center=(0,0), name='A')
p += ER.relation(center=(4,0), name='B')
p += ER.boldarc(names=['A', 'B'])
print(p)
```



41.5 Entity and relation with attributes

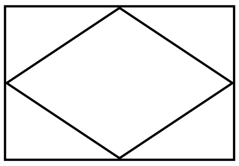
```
from latextool_basic import *
p = Plot()
p += ER.entity(center=(0,0),
               name='Student',
               attribs=['sid', 'name', 'gpa'], keys=['sid'])
ER.attrib_width = 3
p += ER.entity(center=(8,0),
               name='Section',
               attribs=['sectionid', 'instructor'], keys=['sectionid'])
ER.attrib_width = 1.5
p += ER.relation(center=(4,0), name='Registers',
                 attribs=['date'],
                 anchor={'date':'south'})
p += ER.line(names=['Student', 'Registers'])
p += ER.line(names=['Section', 'Registers'])
print(p)
```



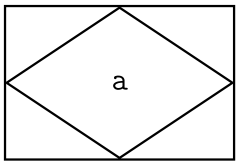
41.6 Weak entity sets and partial keys

41.7 Associative entity

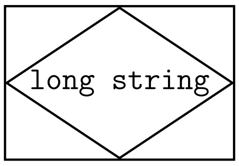
```
from latextool_basic import *
p = Plot()
p += ER.associative_entity_only(center=(0,0), name='a')
print(p)
```



```
from latextool_basic import *
p = Plot()
p += ER.associative_entity_only(center=(0,0), name='a', label='a')
print(p)
```



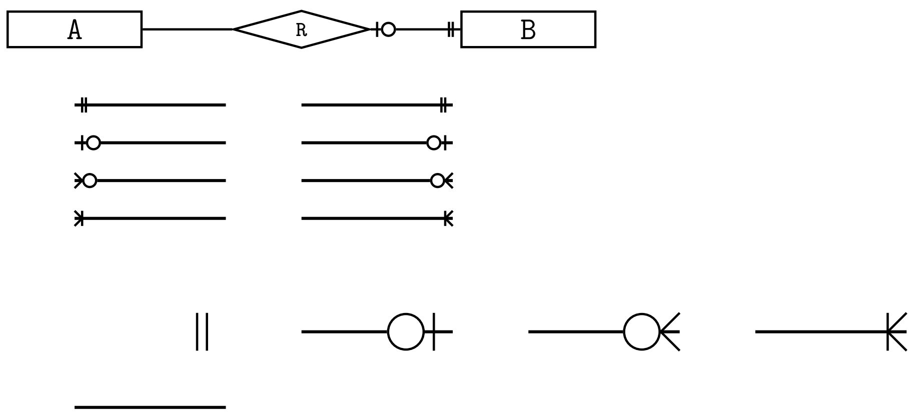
```
from latextool_basic import *
p = Plot()
p += ER.associative_entity_only(center=(0,0), name='a', label='long string'
)
print(p)
```



41.8 Crow's foot

```
from latextool_basic import *
p = Plot()
ER.attrib_width=0.9
ER.attrib_height=0.5
ER.attrib_dist=0.4
ER.attrib_sep=0.4
ER.entity_width=1.8
ER.entity_height=0.5
ER.relation_width=1.8
ER.relation_height=0.5

p += ER.entity(center=(0,0), name='A', attribs=[], keys=[])
p += ER.relation(center=(3,0), name='R', attribs=[], label=r'{\scriptsize R}')
p += ER.entity(center=(6,0), name='B', attribs=[], keys=[])
p += ER.line(names=['B', 'R'])
p += ER.line(names=['A', 'R'])
crowfoot(p, x=3 + 0.9, y=0, kind="0..1", direction="west")
crowfoot(p, x=6 - 0.9, y=0, kind="1", direction="east")



# Test "west"
p += Line(points=[(0,-1),(2, -1)])
crowfoot(p, x=0, y=-1, kind="1", direction="west")

p += Line(points=[(0,-1.5),(2, -1.5)])
crowfoot(p, x=0, y=-1.5, kind="0..1", direction="west")

p += Line(points=[(0,-2),(2, -2)])
crowfoot(p, x=0, y=-2, kind="*", direction="west")

p += Line(points=[(0,-2.5),(2, -2.5)])
crowfoot(p, x=0, y=-2.5, kind="1..*", direction="west")

# Test "east"
p += Line(points=[(3,-1),(5, -1)])
crowfoot(p, x=5, y=-1, kind="1", direction="east")

p += Line(points=[(3,-1.5),(5, -1.5)])
crowfoot(p, x=5, y=-1.5, kind="0..1", direction="east")

p += Line(points=[(3,-2),(5, -2)])
```

```
crowfoot(p, x=5, y=-2, kind="*", direction="east")

p += Line(points=[(3,-2.5),(5, -2.5)])
crowfoot(p, x=5, y=-2.5, kind="1..*", direction="east")

# Test changing dx
x = 0
y = -4
p += Line(points=[(x,-5),(x + 2, -5)])
crowfoot(p, x=x+2, y=y, dx=0.25, kind="1", direction="east")
x += 3
p += Line(points=[(x, y),(x + 2, y)])
crowfoot(p, x=x+2, y=y, dx=0.25, kind="0..1", direction="east")
x += 3
p += Line(points=[(x, y),(x + 2, y)])
crowfoot(p, x=x+2, y=y, dx=0.25, kind="*", direction="east")
x += 3
p += Line(points=[(x, y),(x + 2, y)])
crowfoot(p, x=x+2, y=y, dx=0.25, kind="1..*", direction="east")

print(p)
```



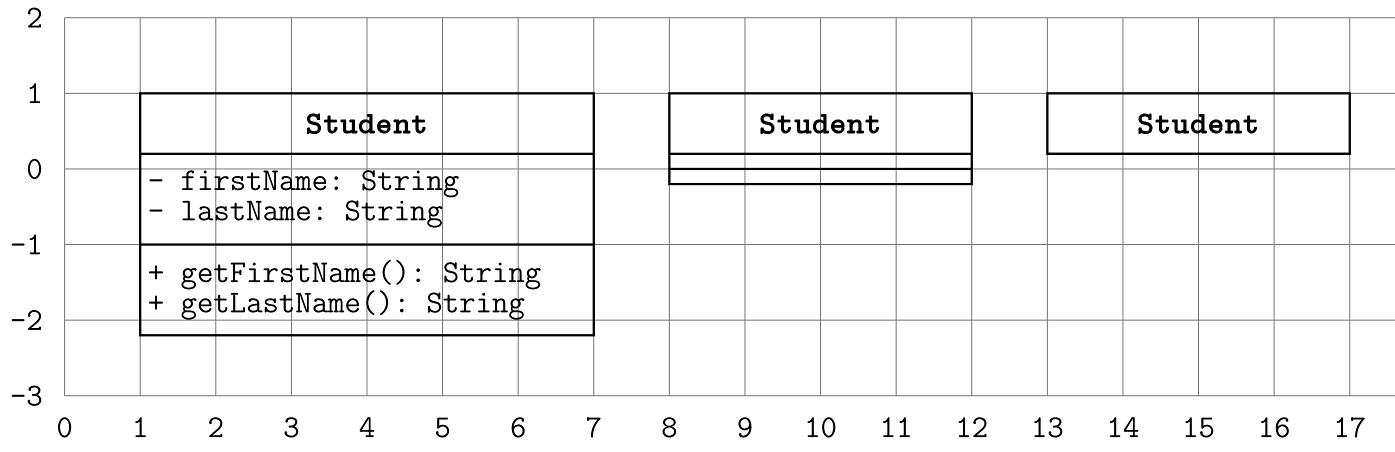
debug: chap-

42 UML diagrams

debug:
uml-classdiag

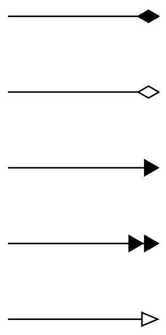
42.1 UML: class diagram

```
from latextool_basic import *
p = Plot()

uml_class(p, 1, 1,
             classname='Student',
             attributes=['- firstName:\ String',
                         '- lastName:\ String'],
             methods=['+ getFirstName():\ String',
                      '+ getLastName():\ String'])

uml_class(p, 8, 1,
             width=4,
             classname='Student')

uml_class(p, 13, 1,
             width=4,
             classname='Student',
             showempty=False)

p += Grid()
print(p)
```



Relevant arrow tips:

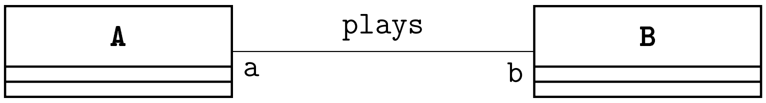
```
from latextool_basic import *
p = Plot()

#p += r'\draw[->>] (1,-5) -- (10,-5);'
#p += r'\draw[->] (1,-5.5) -- (10,-5.5);'
#p += r'\draw[-diamond] (1,-6) -- (10,-6);'
#p += r'\draw[-open diamond] (1,-6.5) -- (10,-6.5);'
#p += r'\draw[-open triangle 45] (1,-7) -- (10,-7);'


p += Line(points=[(0,0),(2,0)], endstyle='diamond', linewidth='0.02')
p += Line(points=[(0,-1),(2,-1)], endstyle='open diamond', linewidth='0.02'
)
p += Line(points=[(0,-2),(2,-2)], endstyle='>', linewidth='0.02')
p += Line(points=[(0,-3),(2,-3)], endstyle='>>', linewidth='0.02')
p += Line(points=[(0,-4),(2,-4)], endstyle='open triangle 45', linewidth='0
.02')

print(p)
```



Associations:

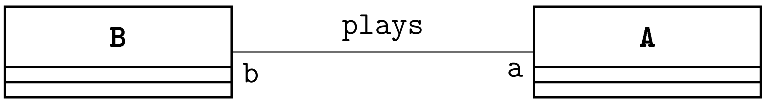
Go right:

```
from latextool_basic import *
p = Plot()
A = uml_class(p, 0, 0, width=3, classname='A')
B = uml_class(p, 7, 0, width=3, classname='B')
association(p, p0=A.right(), p1=B.left(), s='plays', c0='a', c1='b')
print(p)
```



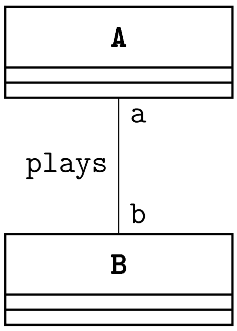
Go left:

```
from latextool_basic import *
p = Plot()
A = uml_class(p, 7, 0, width=3, classname='A')
B = uml_class(p, 0, 0, width=3, classname='B')
association(p, p0=A.left(), p1=B.right(), s='plays', c0='a', c1='b')
print(p)
```



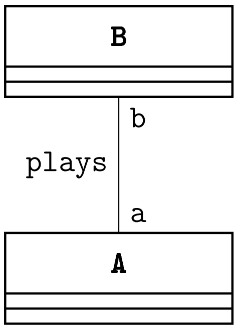
Go down:

```
from latextool_basic import *
p = Plot()
A = uml_class(p, 0, 3, width=3, classname='A')
B = uml_class(p, 0, 0, width=3, classname='B')
association(p, p0=A.bottom(), p1=B.top(), s='plays', c0='a', c1='b')
print(p)
```



Go up:

```
from latextool_basic import *
p = Plot()
A = uml_class(p, 0, -3, width=3, classname='A')
B = uml_class(p, 0, 0, width=3, classname='B')
association(p, p0=A.top(), p1=B.bottom(), s='plays', c0='a', c1='b')
print(p)
```



Go right-up (long-short):

```
from latextool_basic import *
p = Plot()
A = uml_class(p, 0, 0, width=3, classname='A')
B = uml_class(p, 5, 1, width=3, classname='B')
p0 = A.right()
p1 = B.bottom()
association(p, p0=p0, p1=p1, s='plays', c0='a', c1='b',
            moves=[('x',p1),('y',p1)])
print(p)
```

Go right-up (short-long):

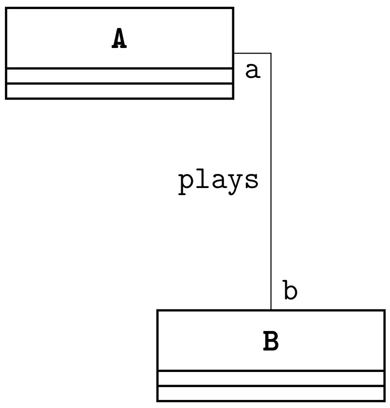
```
from latextool_basic import *
p = Plot()
A = uml_class(p, 0, 0, width=3, classname='A')
B = uml_class(p, 2, 4, width=3, classname='B')
p0 = A.right()
p1 = B.bottom()
association(p, p0=p0, p1=p1, s='plays', c0='a', c1='b',
            moves=[('x',p1),('y',p1)])
print(p)
```

Go right-down (long-short):

```
from latextool_basic import *
p = Plot()
A = uml_class(p, 0, 0, width=3, classname='A')
B = uml_class(p, 5, -1, width=3, classname='B')
p0 = A.right()
p1 = B.top()
association(p, p0=p0, p1=p1, s='plays', c0='a', c1='b',
            moves=[('x',p1),('y',p1)])
print(p)
```

Go right-down (short-long):

```
from latextool_basic import *
p = Plot()
A = uml_class(p, 0, 0, width=3, classname='A')
B = uml_class(p, 2, -4, width=3, classname='B')
p0 = A.right()
p1 = B.top()
association(p, p0=p0, p1=p1, s='plays', c0='a', c1='b',
            moves=[('x',p1),('y',p1)])
print(p)
```

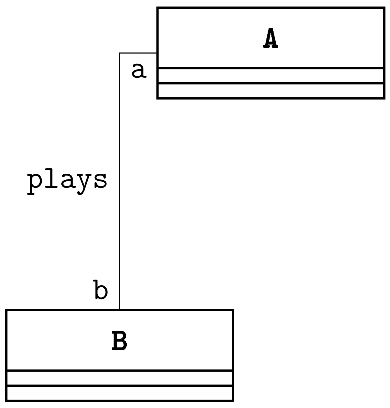


Go left-down (long-short):

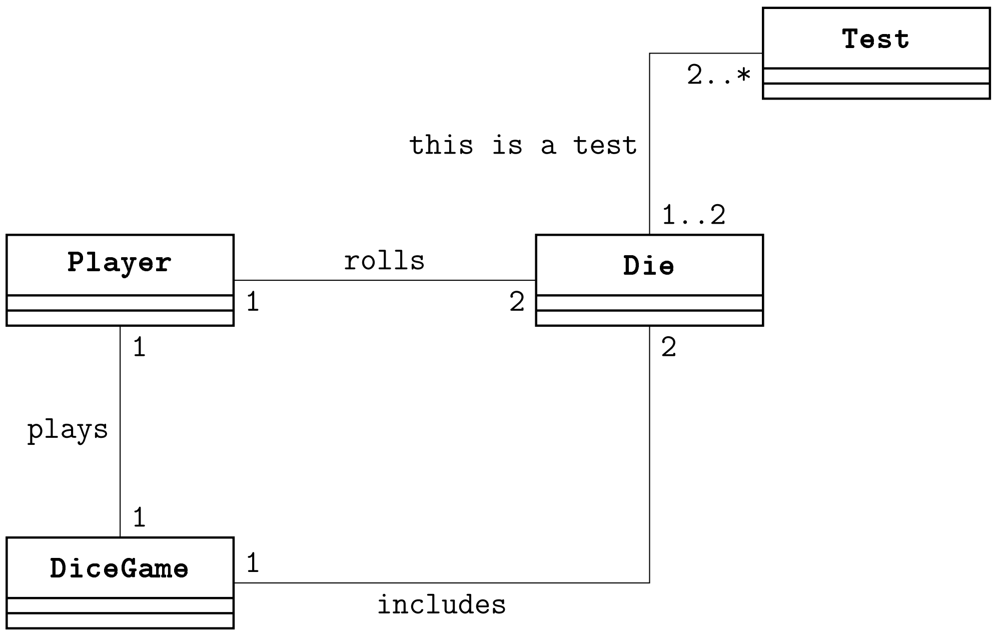
```
from latextool_basic import *
p = Plot()
A = uml_class(p, 0, 0, width=3, classname='A')
B = uml_class(p, -5, -1, width=3, classname='B')
p0 = A.left()
p1 = B.top()
association(p, p0=p0, p1=p1, s='plays', c0='a', c1='b',
            moves=[('x',p1),('y',p1)])
print(p)
```

Go left-down (short-long):

```
from latextool_basic import *
p = Plot()
A = uml_class(p, 0, 0, width=3, classname='A')
B = uml_class(p, -2, -4, width=3, classname='B')
p0 = A.left()
p1 = B.top()
association(p, p0=p0, p1=p1, s='plays', c0='a', c1='b',
            moves=[('x',p1),('y',p1)])
print(p)
```

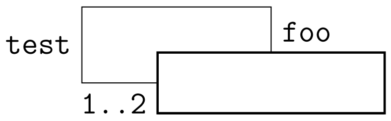


```
from latextool_basic import *
p = Plot()

player = uml_class(p, 0, 0, width=3, classname='Player')
die = uml_class(p, 7, 0, width=3, classname='Die')
dicegame = uml_class(p, 0, -4, width=3, classname='DiceGame')
test = uml_class(p, 10, 3, width=3, classname='Test')

association(p, p0=player.bottom(), p1=dicegame.top(), s='plays',
            c0='1', c1='1', layout='n')
association(p, p0=player.right(), p1=die.left(), s='rolls',
                c0='1', c1='2', layout='e')
association(p, p0=dicegame.right(), p1=die.bottom(), s='includes',
                   c0='1', c1='2', layout='en')
association(p, p0=die.top(), p1=test.left(), s='this is a test',
                   c0='1..2', c1='2..*', layout='ne')
print(p)
```



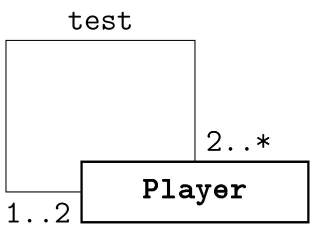
Reflexive associations:

```
from latextool_basic import *
p = Plot()
player = uml_class(p, 0, 0, width=3, classname='', showempty=False)
p0 = player.left()
p1 = player.top()
association(p, p0=p0, p1=p1, s='test',
               c0='1..2', c1='foo', moves=['w','n',('x',p1),('y',p1)])
print(p)
```



```
from latextool_basic import *
p = Plot()
player = uml_class(p, 0, 0, width=3, classname='Player', showempty=False)
p0 = player.right()
p1 = player.top()
association(p, p0=p0, p1=p1, s='test',
            c0='1..2', c1='2..*', moves=['e','n',('x',p1),('y',p1)])
print(p)
```

```
from latextool_basic import *
p = Plot()
player = uml_class(p, 0, 0, width=3, classname='Player', showempty=False)
association(p, p0=player.left(), p1=player.top(), s='test',
               c0='1..2', c1='2..*', layout='wnes',
               dn=2)
print(p)
```



```
from latextool_basic import *
p = Plot()
person = uml_class(p, 0, 0, width=3, classname='Person', showempty=False)
room = uml_class(p, 5, 0, width=3, classname='Room', showempty=False)
knife = uml_class(p, 2.5, -2, width=3, classname='Knife', showempty=False)

association(p, p0=person.right(), p1=room.left(), s='in',
               c0='*', c1='1', layout='e')
association(p, p0=person.left(), p1=person.top(), s='talks to',
               c0='1', c1='1', layout='wnes')

p0 = knife.right()
p1 = room.bottom()
association(p, p0=p0, p1=p1, s='in',
               c0='0', c1='*', moves=[('x',p1),('y',p1)])

p0 = person.bottom()
p1 = knife.left()
association(p, p0=p0, p1=p1, s='holds',
               c0='*', c1='0', moves=[('y', p1), ('x', p1)])

print(p)
```

debug: uml-
sequencediag

42.2 UML: sequence diagram

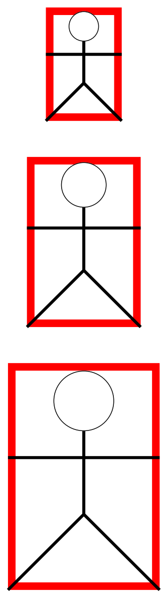
PYTHON ERROR. See source, stderr, stdout below

```
from latextool_basic import *
p = Plot()

klsnames = ['main:Main', ':DiceGame', 'die1:Die', 'die2:Die']

messages = [(':DiceGame','play()'),
            ('die1:Die', 'roll()'),
            ('return',''),
            ('die1:Die', 'getFaceValue()'),
            ('return','faceValue'),
            ('die2:Die', 'roll()'),
            ('return',''),
            ('die2:Die', 'getFaceValue()'),
            (None,''), # no return line
            ('return',''),
            (None,''),
           ]

X,Y = 0,0
dY = -1
width = 2.5
hsep = 1.5
worldlinelength = 5
activationbarwidth = 0.4

def kls(p, x, y, s):
    return uml_class(p, x, y, width=width, classname=s, showempty=False)

k = {}
x = X
for klsname in klsnames:
    k[klsname] = kls(p, x, Y, klsname)
    x += width + hsep

def activationbar(klsname, y0, y1):
    p0 = k[klsname].bottom()
    x0 = p0[0] - activationbarwidth / 2.0
    x1 = p0[0] + activationbarwidth / 2.0
    return Rect(x0=x0, y0=y0, x1=x1, y1=y1, background='white')

Y = k.values()[0].bottom()[1] # start y coordinate at the bottom of the cla
ss
                              # boxes
Y += dY

f = 'main:Main'
stack = [(f, Y)]
activationbars = [] # to be drawn AFTER the timeline
display_return = True

for x in messages:
    if x[0] == 'return':
        Y += dY
        s = x[1]
        g, Y1 = stack.pop()
        if s != '' or display_return:
            a, b = uml_functioncall(k, klsname0=f, klsname1=g, y=Y, s=s,
                   activationbarwidth=activationbarwidth
```

```
Traceback (most recent call last):
  File "pppikubs.tmp.py", line 41, in <module>
    Y = k.values()[0].bottom()[1] # start y coordinate at the bottom of the
 class
TypeError: 'dict_values' object is not subscriptable
```

debug: uml-usecasediagra

42.3 UML: use case diagram



debug: matri

43 Matrix

```
from latextool_basic import *

m = Matrix([[1,2],[3,4]])

print(latex_inverse(m))
```

$$\begin{aligned}\begin{bmatrix} [cc|cc]1 & 2 & 1 & 0 \\ 3 & 4 & 0 & 1 \end{bmatrix} &\xrightarrow{R_2 \to R_2 + (-3)\,R_1} \begin{bmatrix} [cc|cc]1 & 2 & 1 & 0 \\ 0 & -2 & -3 & 1 \end{bmatrix} \\ &\xrightarrow{R_2 \to (-0.5)\,R_2} \begin{bmatrix} [cc|cc]1 & 2 & 1 & 0 \\ -0.0 & 1.0 & 1.5 & -0.5 \end{bmatrix} \\ &\xrightarrow{R_1 \to R_1 + (-2)\,R_2} \begin{bmatrix} [cc|cc]1.0 & 0.0 & -2.0 & 1.0 \\ -0.0 & 1.0 & 1.5 & -0.5 \end{bmatrix}\end{aligned}$$

$$\begin{aligned}\begin{bmatrix} [cc|cc]1 & 2 & 1 & 0 \\ 3 & 4 & 0 & 1 \end{bmatrix} &\xrightarrow{R_2 \to R_2 + (-3)\,R_1} \begin{bmatrix} [cc|cc]1 & 2 & 1 & 0 \\ 0 & -2 & -3 & 1 \end{bmatrix} \\ &\xrightarrow{R_2 \to (-0.5)\,R_2} \begin{bmatrix} [cc|cc]1 & 2 & 1 & 0 \\ -0.0 & 1.0 & 1.5 & -0.5 \end{bmatrix} \\ &\xrightarrow{R_1 \to R_1 + (-2)\,R_2} \begin{bmatrix} [cc|cc]1.0 & 0.0 & -2.0 & 1.0 \\ -0.0 & 1.0 & 1.5 & -0.5 \end{bmatrix}\end{aligned}$$

??? What's wrong with this??? Check again CISS380.

debug: boardgames.t

44 Board games

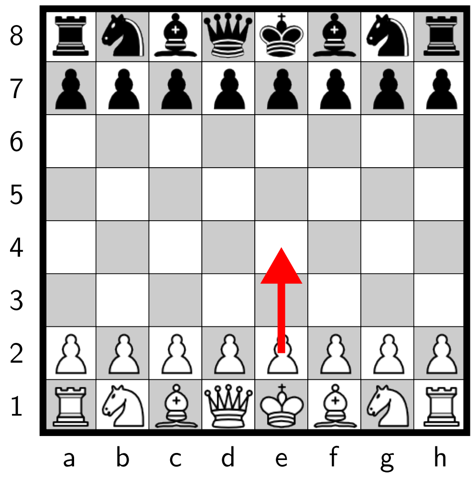
debug: chess.

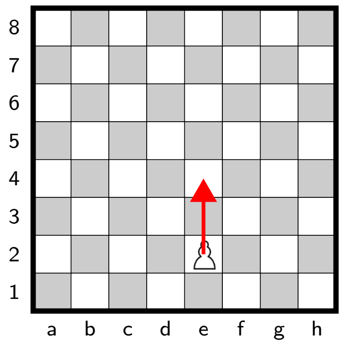
44.1 Chess

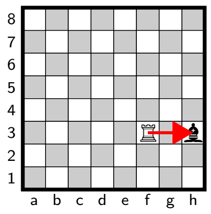
The chess function read image data from image files stored at `data/chess/png/`. The names of the files are `bp.png` for black pawn and `wp.png` for white pawn. Etc. The directory `data/` must be in the same directory as the directory containing `latextool_basic.py`.

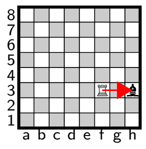
[Note that the chess images for pawns are slightly to the right.]

```
from latextool_basic import *
p = Plot()

xs = [list('rnbqkbnr'),
      list('pppppppp'),
      list('        '),
      list('        '),
      list('        '),
      list('        '),
      list('PPPPPPPP'),
      list('RNBQKBNR'),
      ]
C = chess(p, x=0, y=0, xs=xs)

p += Line(points=[C[6][4].center(), C[4][4].center()], linecolor='red',
          linewidth=0.1, endstyle='>')

print(p)
```





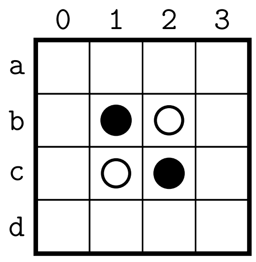




debug: othell

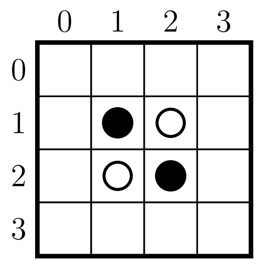
44.2 Othello

```
from latextool_basic import *
p = Plot()

m = [['', '', '',''],
     ['', '@', 'O',''],
     ['', 'O', '@',''],
     ['', '', '','']]
board = othello(p=p, m=m)
print(p)
```



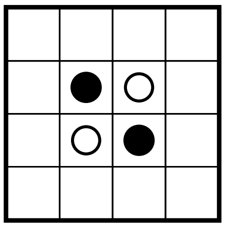
Specify row and column labels:

```
from latextool_basic import *
p = Plot()

m = [['', '', '',''],
     ['', '@', 'O',''],
     ['', 'O', '@',''],
     ['', '', '','']]
board = othello(p=p, m=m,
                rownames=range(4),
                colnames=range(4),
                )
print(p)
```



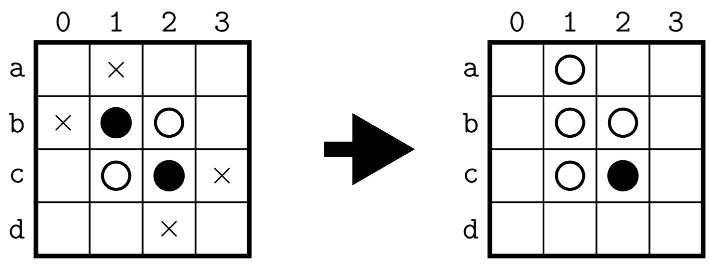
No row, col labels:

```
from latextool_basic import *
p = Plot()

m = [['', '', '',''],
     ['', '@', 'O',''],
     ['', 'O', '@',''],
     ['', '', '','']]
board = othello(p=p, m=m, rownames=[], colnames=[])
print(p)
```



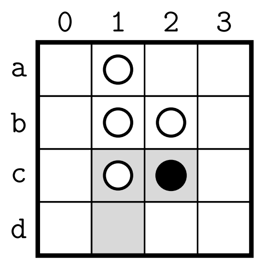
Position:

```
from latextool_basic import *
p = Plot()
X = r'$\times$'
m = [['', X  , '',  ''],
     [X , '@', 'O', ''],
     ['', 'O', '@', X ],
     ['', '' , X ,  '']]
board0 = othello(p=p, m=m, x=0, y=0)
m = [['', 'O', '',  ''],
     ['', 'O', 'O', ''],
     ['', 'O', '@', ''],
     ['', '' , '' , '']]
board1 = othello(p=p, m=m, x=6, y=0)
x0,y0 = board0.right(); x0 += 1; p0 = (x0,y0)
x1,y1 = board1.left(); x1 -= 1; p1 = (x1,y1)
p += Line(points=[p0, p1], linewidth=0.2, endstyle='>')
print(p)
```



Highlighting a cell:

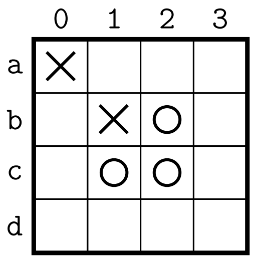
```
from latextool_basic import *
p = Plot()
X = r'$\times$'
m = [['', 'O', '',  ''],
     ['', 'O', 'O', ''],
     ['', 'O', '@', ''],
     ['', '' , '' , '']]
board = othello(p=p, m=m, x=6, y=0, do_not_plot=True)

def highlight(p, row, col):
    x0,y0 = board[row][col].bottomleft()
    x1,y1 = board[row][col].topright()
    p += Rect(x0=x0, y0=y0, x1=x1, y1=y1,
         background='black!15',label='', linewidth=0)

highlight(p, 2, 1)
highlight(p, 3, 1)
highlight(p, 2, 2)

board = othello(p=p, m=m, x=6, y=0)
print(p)
```



debug: ttt.te

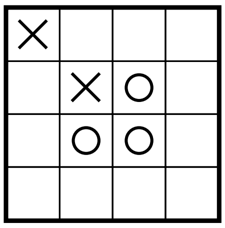
44.3 Tic-tac-toe

```
from latextool_basic import *
p = Plot()

X = 'X'; O = 'O'; s = ''

m = [[X, s, s, s],
     [s, X, O, s],
     [s, O, O, s],
     [s, s, s, s]]
board = ttt(p=p, m=m)
print(p)
```



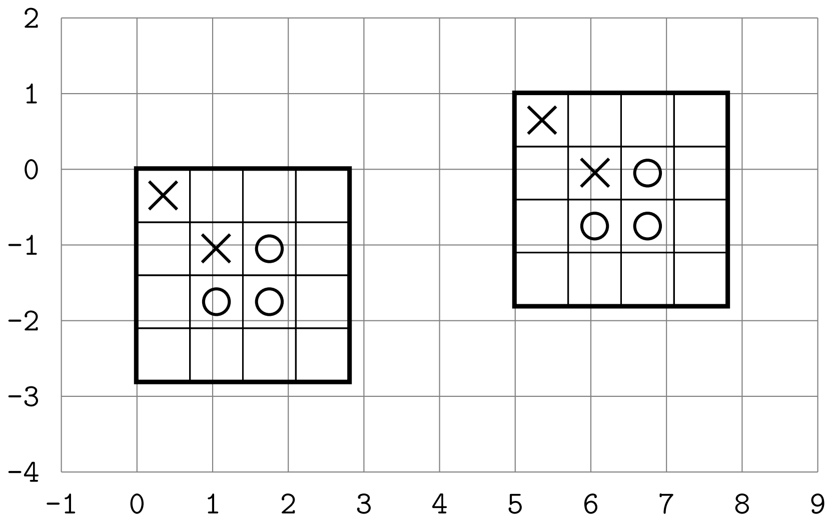
no row and column labels:

```
from latextool_basic import *
p = Plot()

X = 'X'; O = 'O'; s = ''

m = [[X, s, s, s],
     [s, X, O, s],
     [s, O, O, s],
     [s, s, s, s]]
board = ttt(p=p, m=m,
            rownames=['','','',''], colnames=['','','',''])
print(p)
```



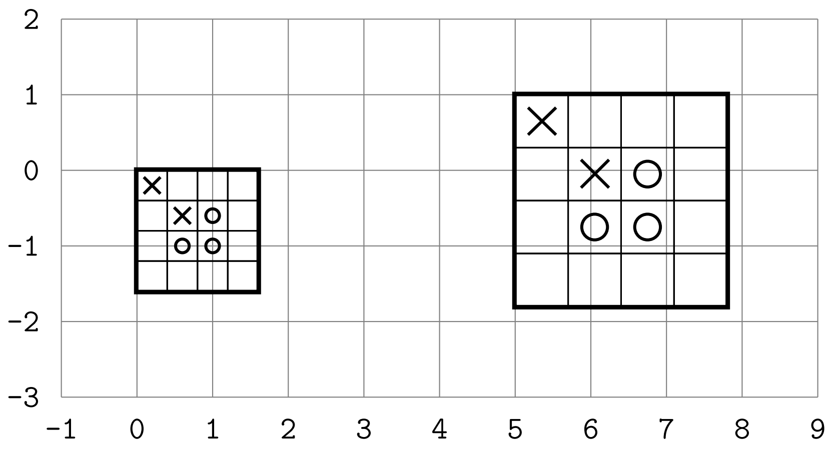
position:

```
from latextool_basic import *
p = Plot()

X = 'X'; O = 'O'; s = ''

m = [[X, s, s, s],
     [s, X, O, s],
     [s, O, O, s],
     [s, s, s, s]]
board = ttt(p=p, x=0, y=0,
            m=m,
            rownames=['','','',''], colnames=['','','',''])
board = ttt(p=p, x=5, y=1,
            m=m,
            rownames=['','','',''], colnames=['','','',''])
p += Grid(x0=-1, y0=-4, x1=9, y1=2)
print(p)
```



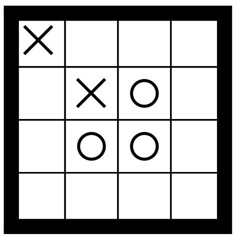
cell size:

```
from latextool_basic import *
p = Plot()

X = 'X'; O = 'O'; s = ''

m = [[X, s, s, s],
     [s, X, O, s],
     [s, O, O, s],
     [s, s, s, s]]
board = ttt(p=p, x=0, y=0,
            m=m,
            rownames=['','','',''], colnames=['','','',''],
            width=0.4)
board = ttt(p=p, x=5, y=1,
            m=m,
            rownames=['','','',''], colnames=['','','',''])
p += Grid(x0=-1, y0=-3, x1=9, y1=2)
print(p)
```



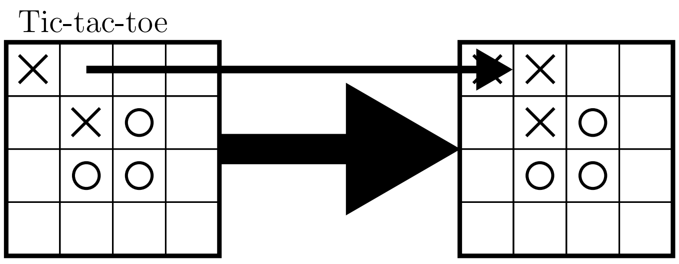
border:

```
from latextool_basic import *
p = Plot()

X = 'X'; O = 'O'; s = ''

m = [[X, s, s, s],
     [s, X, O, s],
     [s, O, O, s],
     [s, s, s, s]]
board = ttt(p=p, x=0, y=0,
            m=m,
            rownames=['','','',''], colnames=['','','',''],
            border_linewidth=0.2)
print(p)
```



using the board

```
from latextool_basic import *
from latexcircuit import *

p = Plot()

X = 'X'; O = 'O'; s = ''

m = [[X, s, s, s],
     [s, X, O, s],
     [s, O, O, s],
     [s, s, s, s]]
board0 = ttt(p=p, x=0, y=0,
            m=m,
            rownames=['','','',''], colnames=['','','',''])
x,y = board0.topleft()
X = POINT(x=x, y=y, r=0, label='Tic-tac-toe', anchor='south west')
p += str(X)

m[0][1] = 'X'
board1 = ttt(p=p, x=6, y=0,
             m=m,
             rownames=['','','',''], colnames=['','','',''])

p += Line(points=[board0.right(), board1.left()], endstyle='>', linewidth=0
.4)

p += Line(points=[board0[0][1].center(),
                  board1[0][1].left()], endstyle='>', linewidth=0.1)

print(p)
```



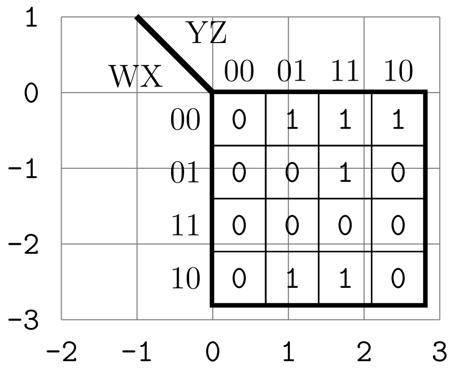
debug: prooft

45 Proof Trees

debug: kmap

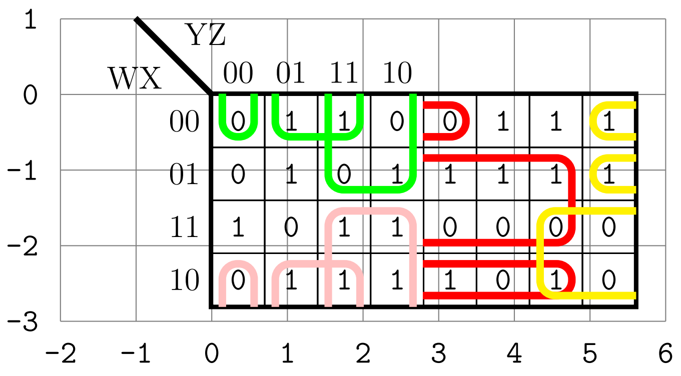
46 K-maps

Drawing closed rect groups. Specify 2 rects: the bottom-left rect and the top-right rect.



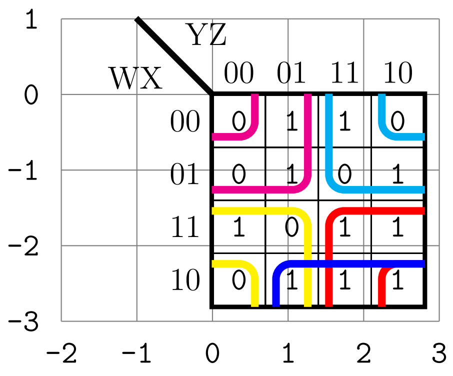
```
from latextool_basic import *

p = Plot()
p += Grid(-2, -3, 3, 1)
m = [[0,1,1,1],
     [0,0,1,0],
     [0,0,0,0],
     [0,1,1,0],
     ]

C = table2(p, m, width=0.7, height=0.7,
           rownames=['00','01','11','10'],
           colnames=['00','01','11','10'],
           rowlabel='WX', collabel='YZ')

#p += kmap_rect(C[3][0], C[2][1], linecolor='red')
#p += kmap_rect(C[1][1], C[0][1], linecolor='blue')
#p += kmap_rect(C[1][0], C[0][0], linecolor='green')
#p += kmap_rect(C[3][2], C[0][2], linecolor='green')
#p += kmap_rect(C[3][0], C[3][3], linecolor='yellow')
#p += kmap_rect(C[0][3], linecolor='cyan')
#p += kmap_rect(C[2][2], C[2][3], linecolor='magenta')

print(p)
```

K-map groups with 1 open side.



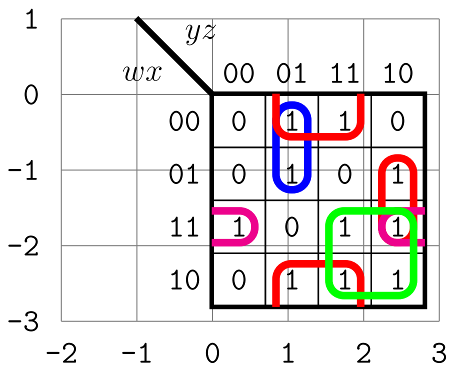
```
from latextool_basic import *

p = Plot()
p += Grid(-2, -3, 6, 1)
m = [[0,1,1,0,0,1,1,1],
     [0,1,0,1,1,1,1,1],
     [1,0,1,1,0,0,0,0],
     [0,1,1,1,1,0,1,0],
     ]

C = table2(p, m, width=0.7, height=0.7,
           rownames=['00','01','11','10'],
           colnames=['00','01','11','10'],
           rowlabel='WX', collabel='YZ')

kmap_WW(p, C[0][4], linecolor='red')
kmap_WW(p, C[2][4], C[1][6], linecolor='red')
kmap_WW(p, C[3][4], C[3][6], linecolor='red')

kmap_EE(p, C[0][7], linecolor='yellow')
kmap_EE(p, C[1][7], C[1][7], linecolor='yellow')
kmap_EE(p, C[3][6], C[2][7], linecolor='yellow')

kmap_SS(p, C[3][0], linecolor='pink')
kmap_SS(p, C[3][1], C[3][2], linecolor='pink')
kmap_SS(p, C[3][2], C[2][3], linecolor='pink')

kmap_NN(p, C[0][0], linecolor='green')
kmap_NN(p, C[0][1], C[0][2], linecolor='green')
kmap_NN(p, C[1][2], C[0][3], linecolor='green')

print(p)
```

K-map groups with two open sides.

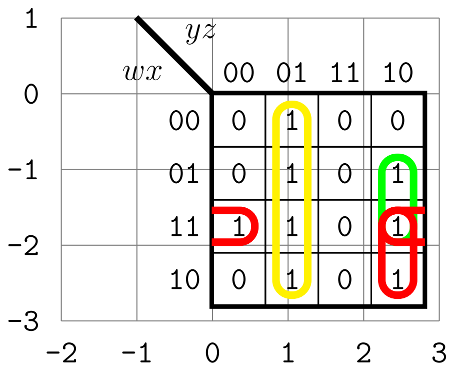


```
from latextool_basic import *

p = Plot()
p += Grid(-2, -3, 3, 1)
m = [[0,1,1,0],
     [0,1,0,1],
     [1,0,1,1],
     [0,1,1,1],
     ]

C = table2(p, m, width=0.7, height=0.7,
           rownames=['00','01','11','10'],
           colnames=['00','01','11','10'],
           rowlabel='WX', collabel='YZ')

kmap_NW(p, C[0][0], linecolor='magenta')
kmap_NW(p, C[1][0], C[0][1], linecolor='magenta')

kmap_NE(p, C[0][3], linecolor='cyan')
kmap_NE(p, C[1][2], C[0][3], linecolor='cyan')

kmap_SW(p, C[3][0], linecolor='yellow')
kmap_SW(p, C[3][0], C[2][1], linecolor='yellow')

kmap_SE(p, C[3][3], linecolor='red')
kmap_SE(p, C[3][2], C[2][3], linecolor='red')
kmap_SE(p, C[3][1], C[3][3], linecolor='blue')

print(p)
```

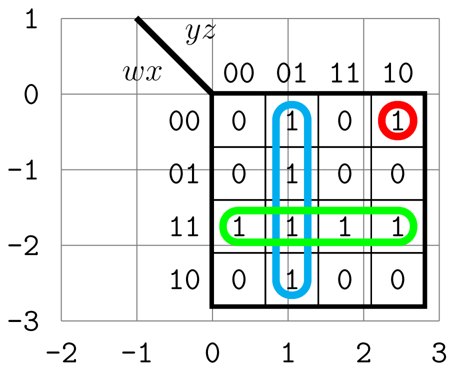
K-maps.



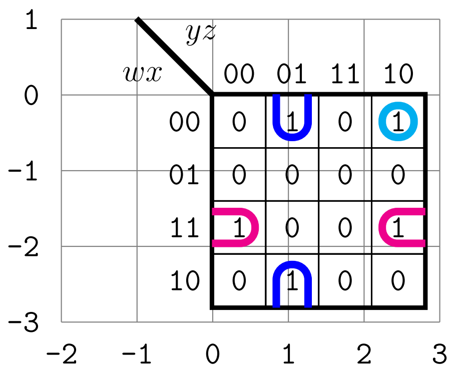
```
from latextool_basic import *

p = Plot()
p += Grid(-2, -3, 3, 1)
m = [[0,1,1,0],
     [0,1,0,1],
     [1,0,1,1],
     [0,1,1,1],
     ]

C = kmap(p, m, width=0.7, height=0.7,
         rowlabel='$wx$', collabel='$yz$')

print(p)
```



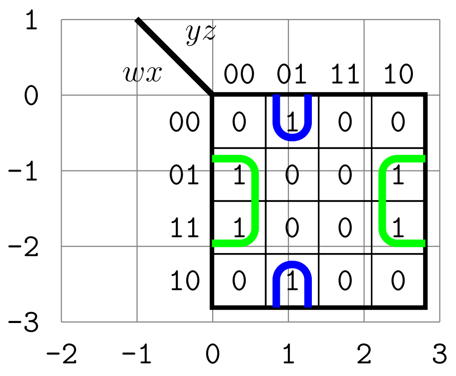
```
from latextool_basic import *

p = Plot()
p += Grid(-2, -3, 3, 1)
m = [[0,1,0,0],
     [0,1,0,1],
     [1,1,0,1],
     [0,1,0,1],
     ]

C = kmap(p, m, width=0.7, height=0.7,
         rowlabel='$wx$', collabel='$yz$')

print(p)
```



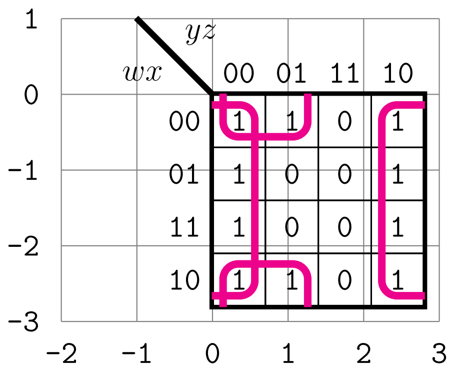
```
from latextool_basic import *

p = Plot()
p += Grid(-2, -3, 3, 1)
m = [[0,1,0,1],
     [0,1,0,0],
     [1,1,1,1],
     [0,1,0,0],
     ]

C = kmap(p, m, width=0.7, height=0.7,
         rowlabel='$wx$', collabel='$yz$')

print(p)
```



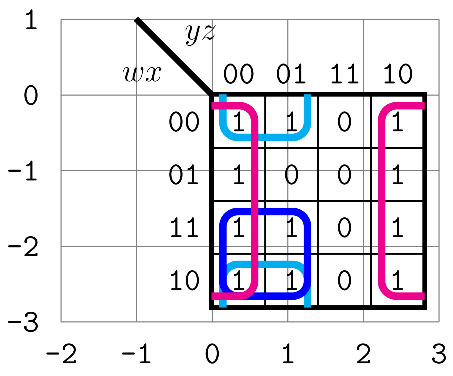
```
from latextool_basic import *

p = Plot()
p += Grid(-2, -3, 3, 1)
m = [[0,1,0,1],
     [0,0,0,0],
     [1,0,0,1],
     [0,1,0,0],
     ]

C = kmap(p, m, width=0.7, height=0.7,
         rowlabel='$wx$', collabel='$yz$')
print(p)
```



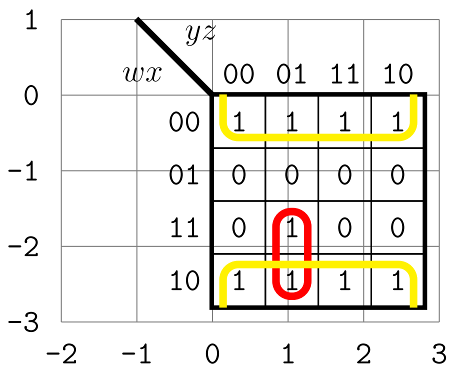
```
from latextool_basic import *

p = Plot()
p += Grid(-2, -3, 3, 1)
m = [[0,1,0,0],
     [1,0,0,1],
     [1,0,0,1],
     [0,1,0,0],
     ]

C = kmap(p, m, width=0.7, height=0.7,
         rowlabel='$wx$', collabel='$yz$')
print(p)
```



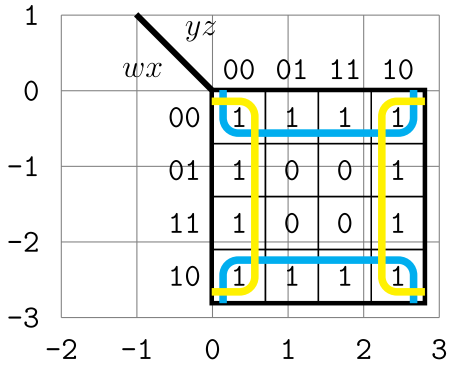
```
from latextool_basic import *

p = Plot()
p += Grid(-2, -3, 3, 1)
m = [[1,1,0,1],
     [1,0,0,1],
     [1,0,0,1],
     [1,1,0,1],
     ]

C = kmap(p, m, width=0.7, height=0.7,
         rowlabel='$wx$', collabel='$yz$')
print(p)
```

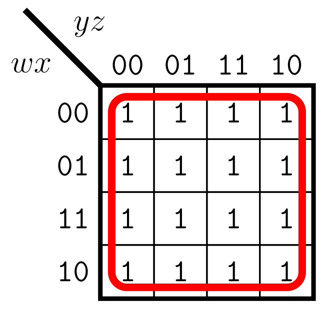


```
from latextool_basic import *

p = Plot()
p += Grid(-2, -3, 3, 1)
m = [[1,1,0,1],
     [1,0,0,1],
     [1,1,0,1],
     [1,1,0,1],
     ]

C = kmap(p, m, width=0.7, height=0.7,
         rowlabel='$wx$', collabel='$yz$')
print(p)
```

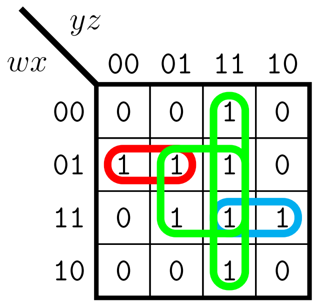


```
from latextool_basic import *

p = Plot()
p += Grid(-2, -3, 3, 1)
m = [[1,1,1,1],
     [0,0,0,0],
     [0,1,0,0],
     [1,1,1,1],
     ]

C = kmap(p, m, width=0.7, height=0.7,
         rowlabel='$wx$', collabel='$yz$')
print(p)
```

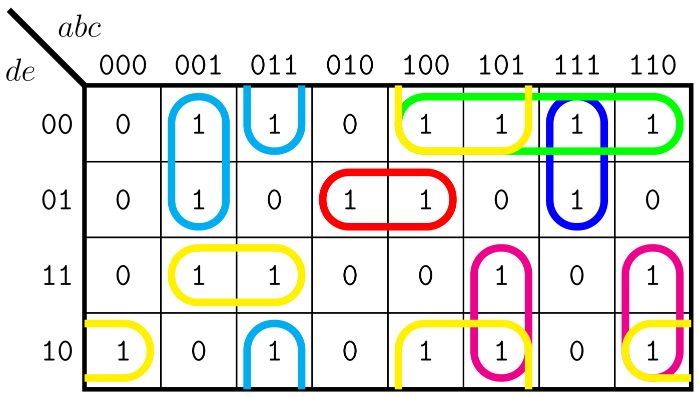




```
from latextool_basic import *

p = Plot()
m = [[1,1,1,1],
     [1,1,1,1],
     [1,1,1,1],
     [1,1,1,1],
     ]

C = kmap(p, m, width=0.7, height=0.7,
         rowlabel='$wx$', collabel='$yz$')
print(p)
```

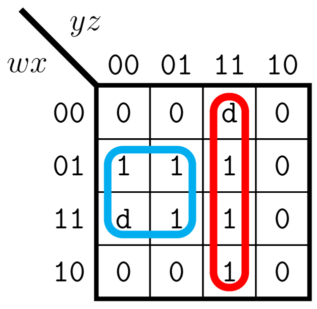


```
from latextool_basic import *

p = Plot()
m = [[0,0,1,0],
     [1,1,1,0],
     [0,1,1,1],
     [0,0,1,0],
     ]

C = kmap(p, m, width=0.7, height=0.7,
         rowlabel='$wx$', collabel='$yz$')
print(p)
```

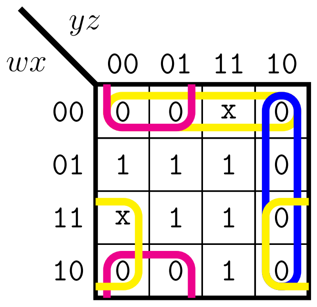


```
from latextool_basic import *

p = Plot()
#p += Grid(-2, -3, 3, 1)
m = [[0,1,1,0,1,1,1,1],
     [0,1,0,1,1,0,1,0],
     [0,1,1,0,0,1,0,1],
     [1,0,1,0,1,1,0,1],
     ]

C = kmap(p, m, width=1, height=1,
         rowlabel='$de$', collabel='$abc$')
print(p)
```

46.1 kmap: specify term(s) to circle

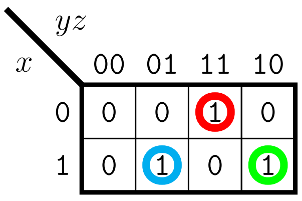


```
from latextool_basic import *

p = Plot()
m = [[0,0,'d',0],
     [1,1,1,0],
     ['d',1,1,0],
     [0,0,1,0],
     ]

C = kmap(p, m, width=0.7, height=0.7,
         circle_terms=[1, 'd'],
         rowlabel='$wx$', collabel='$yz$')
print(p)
```

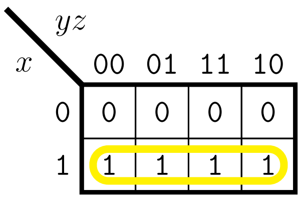


```
from latextool_basic import *

p = Plot()
m = [[0,0,'x',0],
     [1,1,1,0],
     ['x',1,1,0],
     [0,0,1,0],
     ]

C = kmap(p, m, width=0.7, height=0.7,
         circle_terms=[0, 'x'],
         rowlabel='$wx$', collabel='$yz$')
print(p)
```

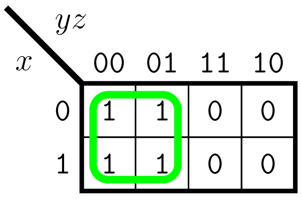
46.2 kmap: 2-by-4



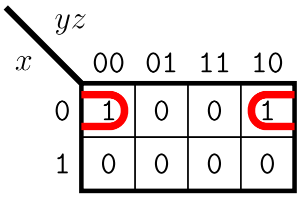
```
from latextool_basic import *

p = Plot()
m = [[0,0,1,0],
     [0,1,0,1],
     ]

C = kmap(p, m, width=0.7, height=0.7,
         rowlabel='$x$', collabel='$yz$')

print(p)
```



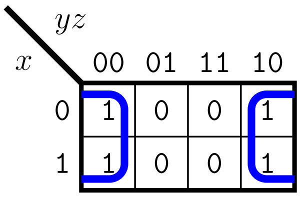
```
from latextool_basic import *

p = Plot()
m = [[0,0,0,0],
     [1,1,1,1],
     ]

C = kmap(p, m, width=0.7, height=0.7,
         rowlabel='$x$', collabel='$yz$')

print(p)
```



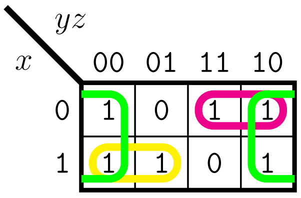
```
from latextool_basic import *

p = Plot()
m = [[1,1,0,0],
     [1,1,0,0],
     ]

C = kmap(p, m, width=0.7, height=0.7,
         rowlabel='$x$', collabel='$yz$')

print(p)
```



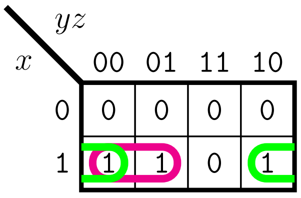
```
from latextool_basic import *

p = Plot()
m = [[1,0,0,1],
     [0,0,0,0],
     ]

C = kmap(p, m, width=0.7, height=0.7,
         rowlabel='$x$', collabel='$yz$')

print(p)
```



```
from latextool_basic import *

p = Plot()
m = [[1,0,0,1],
     [1,0,0,1],
     ]

C = kmap(p, m, width=0.7, height=0.7,
         rowlabel='$x$', collabel='$yz$')

print(p)
```



```
from latextool_basic import *

p = Plot()
m = [[1,0,1,1],
     [1,1,0,1],
     ]

C = kmap(p, m, width=0.7, height=0.7,
         rowlabel='$x$', collabel='$yz$')

print(p)
```

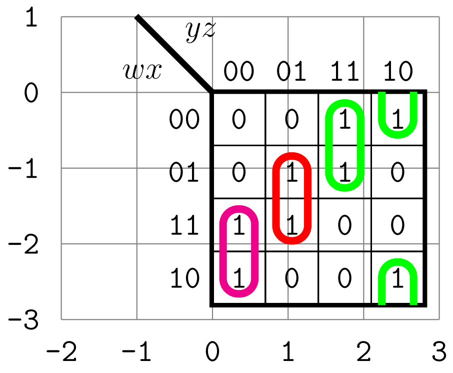


```
from latextool_basic import *

p = Plot()
m = [[0,0,0,0],
     [1,1,0,1],
     ]

C = kmap(p, m, width=0.7, height=0.7,
         rowlabel='$x$', collabel='$yz$')

print(p)
```

46.3 kmap: force choice of prime implicant

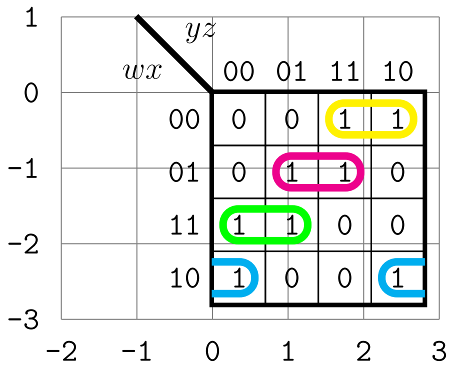
There is more than one way to select essential prime implicants.

Default:



```
from latextool_basic import *

p = Plot()
p += Grid(-2, -3, 3, 1)
m = [[0,0,1,1],
     [0,1,1,0],
     [1,1,0,0],
     [1,0,0,1],
     ]

C = kmap(p, m, width=0.7, height=0.7,
         rowlabel='$wx$', collabel='$yz$')
print(p)
```

You can force "do not remove $((0, 0), (2, 3))$" using `donotremove` parameter:



```
from latextool_basic import *

p = Plot()
p += Grid(-2, -3, 3, 1)
m = [[0,0,1,1],
     [0,1,1,0],
     [1,1,0,0],
     [1,0,0,1],
     ]

C = kmap(p, m, width=0.7, height=0.7,
         rowlabel='$wx$', collabel='$yz$',
         donotremove=[((0,0),(2,3))])
print(p)
```

46.4 kmap: style selector

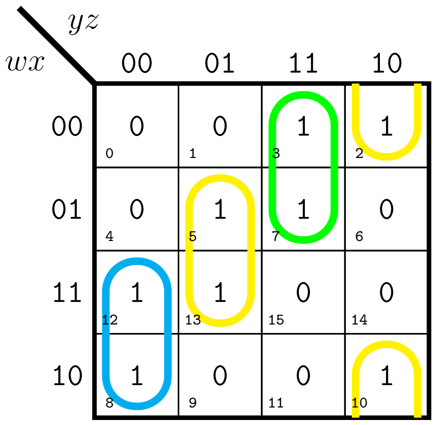
You can select color and linewidth for the implicants:

46.5 kmap: change distance of implicant boundary to rect

`d = 0.1`:

```
d = 0.3
```

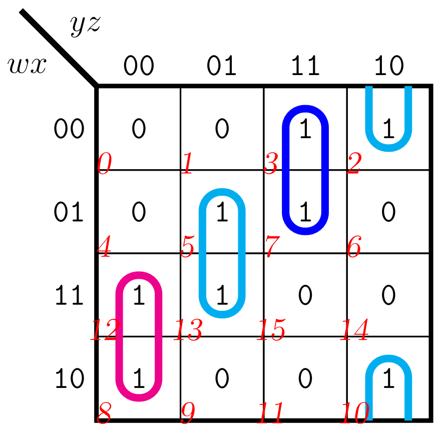
46.6 kmap: decimal notation



```
from latextool_basic import *

p = Plot()
m = [[0,0,1,1],
     [0,1,1,0],
     [1,1,0,0],
     [1,0,0,1],
     ]

C = kmap(p, m, width=1.1, height=1.1,
         rowlabel='$wx$', collabel='$yz$',
         decimal=True
         )

print(p)
```

46.7 kmap: decimal notation 2

You can specify offset of decimal notation and the formatting string:



```
from latextool_basic import *

p = Plot()
m = [[0,0,1,1],
     [0,1,1,0],
     [1,1,0,0],
     [1,0,0,1],
     ]

C = kmap(p, m, width=1.1, height=1.1,
         rowlabel='$wx$', collabel='$yz$',
         d = 0.3,
         decimal=True,
         decimal_offset=0.1,
         decimal_format=r'{\textred{\textsl{%s}}}'
         )

print(p)
```

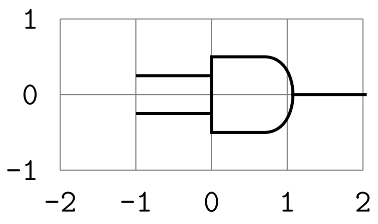
46.8 kmap: using coordinates of rects in the kmap cells

I use a slightly large `d` value so that the implicant boundaries avoid the decimal notation.

debug:
logic-design.t

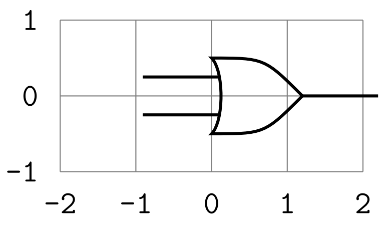
47 Logic design

47.1 AND gate



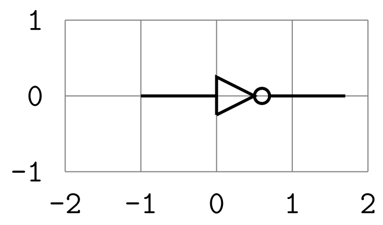
```
from latextool_basic import *
from latexcircuit import *

p = Plot()
g = AND_GATE(x=0, y=0, inputs=2)
p += str(g)

for x,y in g.inputs():
    p += Line(points=[(x - 1, y), (x, y)])

x,y = g.output()
p += Line(points=[(x, y), (x + 1,y)])

p += Grid(x0=-2, y0=-1, x1=2, y1=1)
print(p)
```

47.2 OR gate



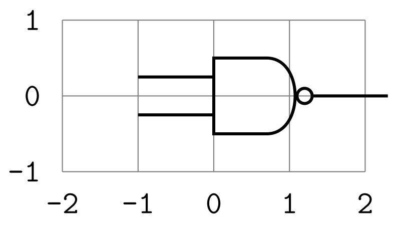
```
from latextool_basic import *
from latexcircuit import *

p = Plot()
g = OR_GATE()
p += str(g)

for x,y in g.inputs():
    p += Line(points=[(x - 1, y), (x, y)])

x,y = g.output()
p += Line(points=[(x, y),(x+1,y)])

p += Grid(x0=-2, y0=-1, x1=2, y1=1)
print(p)
```

47.3 NOT gate



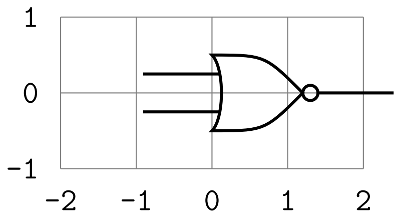
```
from latextool_basic import *
from latexcircuit import *

p = Plot()
g = NOT_GATE()
p += str(g)

x,y = g.input()
p += Line(points=[(x,y),(x - 1,y)])

x,y = g.output()
p += Line(points=[(x,y),(x + 1,y)])

p += Grid(x0=-2, y0=-1, x1=2, y1=1)
print(p)
```

47.4 NAND gate



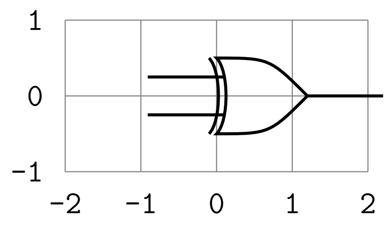
```
from latextool_basic import *
from latexcircuit import *

p = Plot()
g = NAND_GATE()
p += str(g)

for x,y in g.inputs():
    p += Line(points=[(x-1,y),(x,y)])

x,y = g.output()
p += Line(points=[(x,y),(x+1,y)])

p += Grid(x0=-2, y0=-1, x1=2, y1=1)
print(p)
```

47.5 NOR gate



```
from latextool_basic import *
from latexcircuit import *

p = Plot()
g = NOR_GATE()
p += str(g)

for x,y in g.inputs():
    p += Line(points=[(x-1,y),(x,y)])

x,y = g.output()
p += Line(points=[(x,y),(x+1,y)])

p += Grid(x0=-2, y0=-1, x1=2, y1=1)
print(p)
```

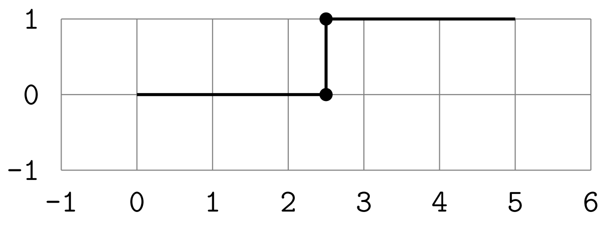
47.6 XOR gate



```
from latextool_basic import *
from latexcircuit import *

p = Plot()
g = XOR_GATE()
p += str(g)

for x,y in g.inputs():
    p += Line(points=[(x-1,y),(x,y)])

x,y = g.output()
p += Line(points=[(x,y),(x+1,y)])

p += Grid(x0=-2, y0=-1, x1=2, y1=1)
print(p)
```

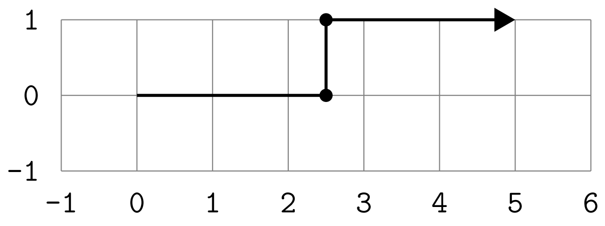
47.7 Orthogonal paths

For drawing orthogonal paths between two points.



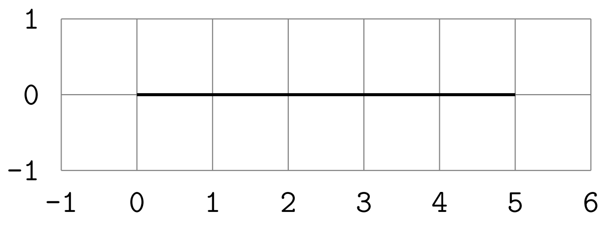
```
from latextool_basic import *
from latexcircuit import *
p = Plot()
p += str(OrthogonalPath([(0,0), (5,1)]))
p += Grid(x0=-1, y0=-1, x1=6, y1=1)
print(p)
```

Orthogonal path with arrow head:



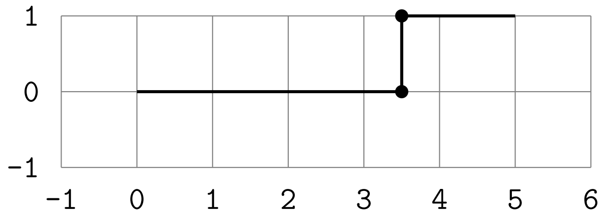
```
from latextool_basic import *
from latexcircuit import *
p = Plot()
p += str(OrthogonalPath([(0,0), (5,1)], endstyle='>', arrowstyle='triangle'
))
p += Grid(x0=-1, y0=-1, x1=6, y1=1)
print(p)
```

Orthogonal paths – no bend case:



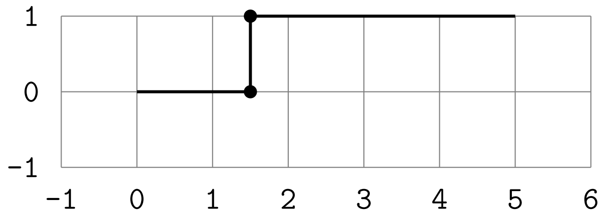
```
from latextool_basic import *
from latexcircuit import *
p = Plot()
p += str(OrthogonalPath([(0,0), (5,0)]))
p += Grid(x0=-1, y0=-1, x1=6, y1=1)
print(p)
```

with a positive horizontal shift of the bend:



```
from latextool_basic import *
from latexcircuit import *
p = Plot()
p += str(OrthogonalPath([(0,0), (5,1)], shifts=[1]))
p += Grid(x0=-1, y0=-1, x1=6, y1=1)
print(p)
```

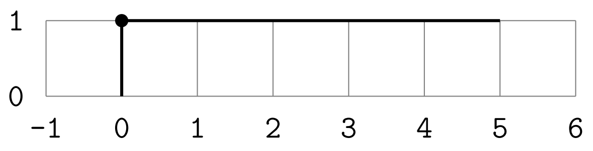
with a negative horizontal shift of the bend:



```
from latextool_basic import *
from latexcircuit import *
p = Plot()
p += str(OrthogonalPath([(0,0), (5,1)], shifts=[-1]))
p += Grid(x0=-1, y0=-1, x1=6, y1=1)
print(p)
```

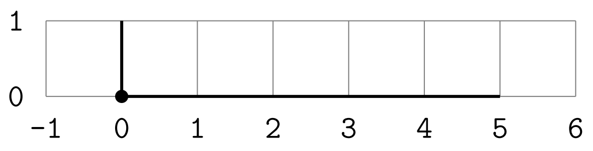
TODO: shifts for vertical case

orthogonal path, direction='vh' (vertical-horizontal):



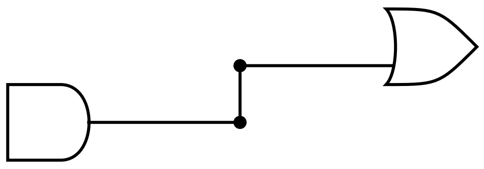
```
from latextool_basic import *
from latexcircuit import *
p = Plot()
p += str(OrthogonalPath(points=[(0,0), (5,1)], direction='vh'))
p += Grid(x0=-1, y0=0, x1=6, y1=1)
print(p)
```

orthogonal path, direction='vh':



```
from latextool_basic import *
from latexcircuit import *
p = Plot()
p += str(OrthogonalPath(points=[(0,1), (5,0)], direction='vh'))
p += Grid(x0=-1, y0=0, x1=6, y1=1)
print(p)
```

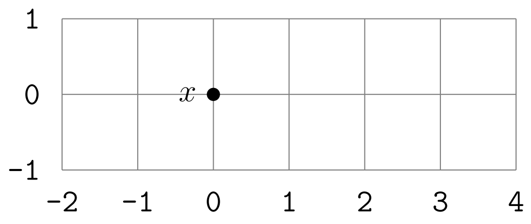
47.8 Orthogonal path with gates



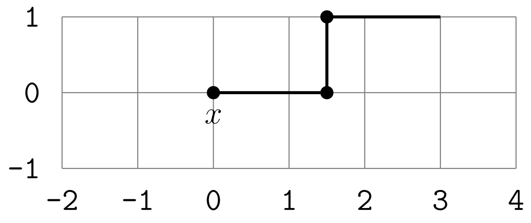
```
from latextool_basic import *
from latexcircuit import *
p = Plot()
AND = AND_GATE(x=0, y=0); p += str(AND)
OR = OR_GATE(x=5, y=1); p += str(OR)
opath = OrthogonalPath(gate0=AND, gate1=OR, input_index=0)
p += str(opath)
print(p)
```

47.9 POINT

POINT:

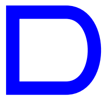


```
from latextool_basic import *
from latexcircuit import *
p = Plot()
X = POINT(x=0, y=0, label='$x$')
p += str(X)
p += Grid(x0=-2, y0=-1, x1=4, y1=1)
print(p)
```



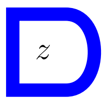
```
from latextool_basic import *
from latexcircuit import *
p = Plot()
X = POINT(x=0, y=0, label='$x$', anchor='north')
p += str(X)
p += str(OrthogonalPath([X.output(), (3,1)]))
p += Grid(x0=-2, y0=-1, x1=4, y1=1)
print(p)
```

47.10 Linecolor for gate

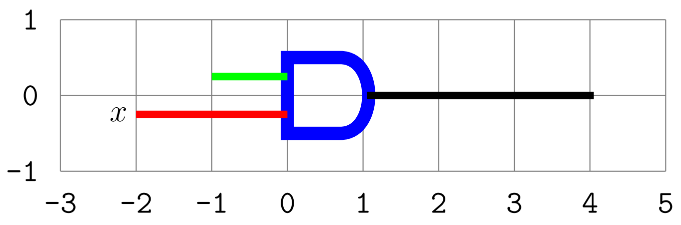


```
from latextool_basic import *
from latexcircuit import *

p = Plot()
g0 = AND_GATE(linewidth=5, linecolor='blue')
p += str(g0)

print(p)
```

47.11 Label for gate

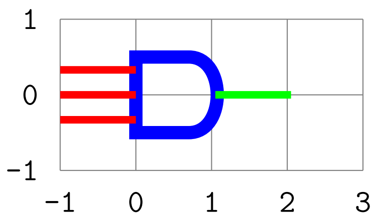


Test 2 inputs and output



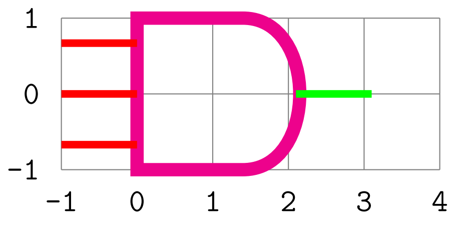
```
from latextool_basic import *
from latexcircuit import *

p = Plot()
g0 = AND_GATE(linewidth=5, linecolor='blue'); p += str(g0)

inputs = g0.inputs()

# Test input 0
x0,y0 = inputs[0]; p0 = (x0, y0)
x1,y1 = x0 - 2, y0; p1 = (x1, y1)
p += Line(points=[p0,p1], linewidth=0.1, linecolor='red')
x0 = x1 - 0.3
p += Rect(x0=x0, y0=y0, x1=x0, y1=y0, linewidth=0, s = '$x$')

# test input 1
x0,y0 = inputs[1]; p0 = (x0, y0)
x1,y1 = x0 - 1, y0; p1 = (x1, y1)
p += Line(points=[p0,p1], linewidth=0.1, linecolor='green')

# test output
x0,y0 = g0.output(); p0 = x0,y0
x1,y1 = x0 + 3, y0; p1 = x1,y1
p += Line(points=[p0,p1], linewidth=0.1, linecolor='black')

p += Grid()
print(p)
```

Test 3 inputs and output



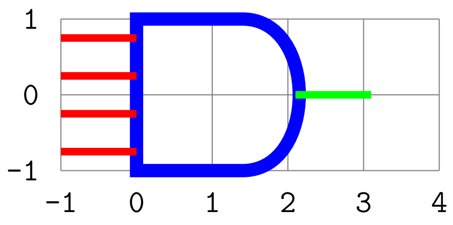
```
from latextool_basic import *
from latexcircuit import *

p = Plot()
g0 = AND_GATE(linewidth=5, linecolor='blue', inputs=3)
p += str(g0)

for x0,y0 in g0.inputs():
    p0 = (x0, y0)
    x1,y1 = x0 - 1, y0
    p1 = (x1, y1)
    p += Line(points=[p0,p1], linewidth=0.1, linecolor='red')

x0,y0 = g0.output()
p0 = (x0, y0)
x1,y1 = x0 + 1, y0
p1 = (x1, y1)
p += Line(points=[p0,p1], linewidth=0.1, linecolor='green')

p += Grid()
print(p)
```

Test different size (height = 2)



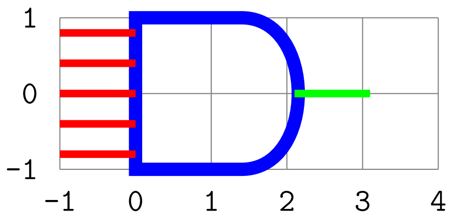
```
from latextool_basic import *
from latexcircuit import *

p = Plot()
g0 = AND_GATE(x=0, y=0, h=2,
              linewidth=5, linecolor='magenta', inputs=3)
p += str(g0)

for x0,y0 in g0.inputs():
    p0 = (x0, y0)
    x1,y1 = x0 - 1, y0
    p1 = (x1, y1)
    p += Line(points=[p0,p1], linewidth=0.1, linecolor='red')

x0,y0 = g0.output()
p0 = (x0, y0)
x1,y1 = x0 + 1, y0
p1 = (x1, y1)
p += Line(points=[p0,p1], linewidth=0.1, linecolor='green')

p += Grid()
print(p)
```

Test height = 2, 4 inputs:



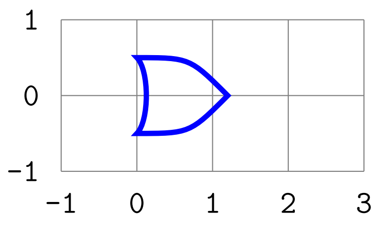
```
from latextool_basic import *
from latexcircuit import *

p = Plot()
g0 = AND_GATE(x=0, y=0, h=2,
              linewidth=5, linecolor='blue', inputs=4)
p += str(g0)

for x0,y0 in g0.inputs():
    p0 = (x0, y0)
    x1,y1 = x0 - 1, y0
    p1 = (x1, y1)
    p += Line(points=[p0,p1], linewidth=0.1, linecolor='red')

x0,y0 = g0.output()
p0 = (x0, y0)
x1,y1 = x0 + 1, y0
p1 = (x1, y1)
p += Line(points=[p0,p1], linewidth=0.1, linecolor='green')

p += Grid()
print(p)
```

Test height = 2, 5 inputs:

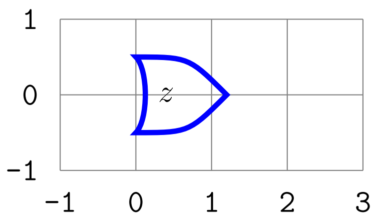


```
from latextool_basic import *
from latexcircuit import *

p = Plot()
g0 = AND_GATE(x=0, y=0, h=2,
              linewidth=5, linecolor='blue', inputs=5)
p += str(g0)

for x0,y0 in g0.inputs():
    p0 = (x0, y0)
    x1,y1 = x0 - 1, y0
    p1 = (x1, y1)
    p += Line(points=[p0,p1], linewidth=0.1, linecolor='red')

x0,y0 = g0.output()
p0 = (x0, y0)
x1,y1 = x0 + 1, y0
p1 = (x1, y1)
p += Line(points=[p0,p1], linewidth=0.1, linecolor='green')

p += Grid()
print(p)
```

OR gate:

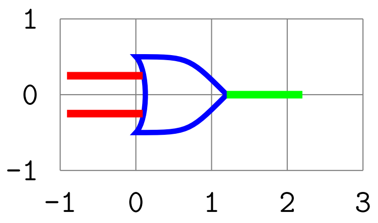


```
from latextool_basic import *
from latexcircuit import *

p = Plot()
g0 = OR_GATE(linewidth=2, linecolor='blue')
p += str(g0)

p += Grid(x0=-1,y0=-1,x1=3,y1=1)

print(p)
```



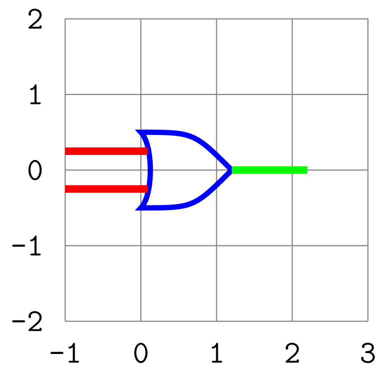
```
from latextool_basic import *
from latexcircuit import *

p = Plot()
g0 = OR_GATE(linewidth=2, linecolor='blue', label='$z$')
p += str(g0)

p += Grid(x0=-1,y0=-1,x1=3,y1=1)

print(p)
```

OR gate testing 2 inputs



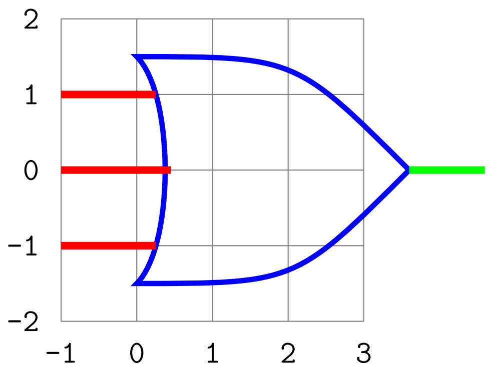
```
from latextool_basic import *
from latexcircuit import *

p = Plot()
g0 = OR_GATE(linewidth=2, linecolor='blue')
p += str(g0)

for x0,y0 in g0.inputs():
    p0 = (x0, y0)
    x1,y1 = x0 - 1, y0
    p1 = (x1, y1)
    p += Line(points=[p0,p1], linewidth=0.1, linecolor='red')

x0,y0 = g0.output()
p0 = (x0, y0)
x1,y1 = x0 + 1, y0
p1 = (x1, y1)
p += Line(points=[p0,p1], linewidth=0.1, linecolor='green')

p += Grid(x0=-1,y0=-1,x1=3,y1=1)

print(p)
```

OR gate testing 2 inputs



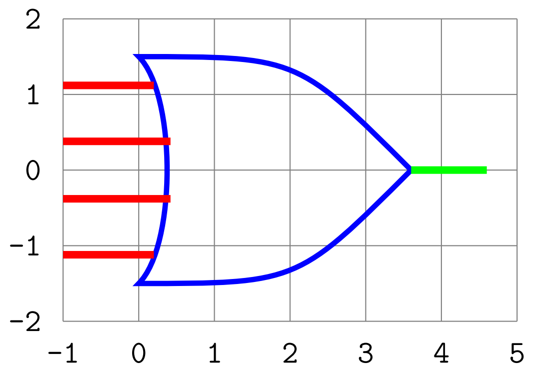
```
from latextool_basic import *
from latexcircuit import *

p = Plot()
g0 = OR_GATE(linewidth=2, x=0, y=0, linecolor='blue')
p += str(g0)

for x0,y0 in g0.inputs():
    p0 = (x0, y0)
    x1, y1 = - 1, y0
    p1 = (x1, y1)
    p += Line(points=[p0,p1], linewidth=0.1, linecolor='red')

x0, y0 = g0.output(); p0 = (x0, y0)
x1, y1 = x0 + 1, y0; p1 = (x1, y1)
p += Line(points=[p0,p1], linewidth=0.1, linecolor='green')

p += Grid(x0=-1,y0=-2,x1=3,y1=2)

print(p)
```

OR gate testing 3 inputs



```
from latextool_basic import *
from latexcircuit import *

p = Plot()
g0 = OR_GATE(linewidth=2, x=0,y=0,h=3, inputs=3, linecolor='blue')
p += str(g0)

for x0,y0 in g0.inputs():
    p0 = (x0, y0)
    p1 = (-1, y0)
    p += Line(points=[p0,p1], linewidth=0.1, linecolor='red')

x0,y0 = g0.output(); p0 = (x0, y0)
x1,y1 = x0 + 1, y0; p1 = (x1, y1)
p += Line(points=[p0, p1], linewidth=0.1, linecolor='green')

p += Grid(x0=-1,y0=-2,x1=3,y1=2)

print(p)
```

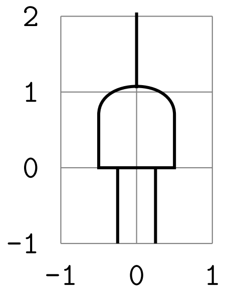
OR gate testing 4 inputs



```
from latextool_basic import *
from latexcircuit import *

p = Plot()
g0 = OR_GATE(linewidth=2, x=0,y=0,h=3, inputs=4,linecolor='blue')
p += str(g0)

for x0,y0 in g0.inputs():
    p0 = (x0, y0); x1, y1 = -1, y0; p1 = (x1, y1)
    p += Line(points=[p0,p1], linewidth=0.1, linecolor='red')

x0,y0 = g0.output()
p0 = (x0, y0); p1 = (x0 + 1, y0)
p += Line(points=[p0,p1], linewidth=0.1, linecolor='green')

p += Grid(x0=-1, y0=-2, x1=5, y1=2)

print(p)
```

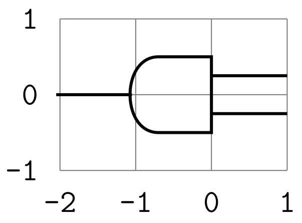
47.12 Rotation

AND gate rotate by $\pi/4$:



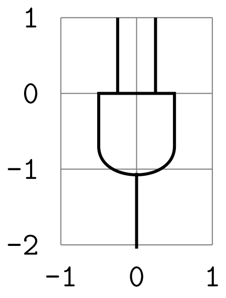
```
from math import pi
from latextool_basic import *
from latexcircuit import *
p = Plot()
g = AND_GATE(x=0, y=0, inputs=2, angle=pi/2)
p += str(g)

for x,y in g.inputs():
    p += Line(points=[(x, y - 1), (x, y)])

x,y = g.output()
p += Line(points=[(x, y),(x, y + 1)])

p += Grid(x0=-1, y0=-1, x1=1, y1=2)
print(p)
```

AND gate rotate by π:



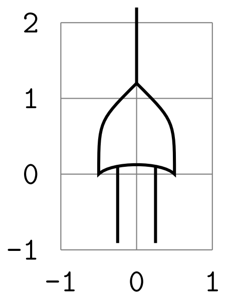
```
from math import pi
from latextool_basic import *
from latexcircuit import *
p = Plot()
g = AND_GATE(x=0, y=0, inputs=2, angle=pi)
p += str(g)

for x,y in g.inputs():
    p += Line(points=[(x + 1, y), (x, y)])

x,y = g.output()
p += Line(points=[(x, y),(x - 1, y)])

p += Grid(x0=-2, y0=-1, x1=1, y1=1)
print(p)
```

AND gate rotate by $3\pi/2$:



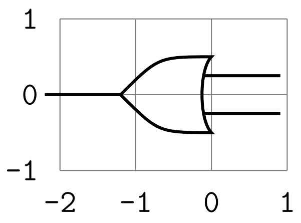
```
from math import pi
from latextool_basic import *
from latexcircuit import *
p = Plot()
g = AND_GATE(x=0, y=0, inputs=2, angle=3*pi/2)
p += str(g)

for x,y in g.inputs():
    p += Line(points=[(x, y + 1), (x, y)])

x,y = g.output()
p += Line(points=[(x, y),(x, y - 1)])

p += Grid(x0=-1, y0=-2, x1=1, y1=1)
print(p)
```

OR gate rotate by $\pi/4$:



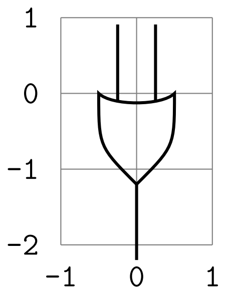
```
from math import pi
from latextool_basic import *
from latexcircuit import *
p = Plot()

g = OR_GATE(x=0, y=0, inputs=2, angle=pi/2)
p += str(g)

for x,y in g.inputs():
    p += Line(points=[(x, y - 1), (x, y)])

x,y = g.output()
p += Line(points=[(x, y),(x, y + 1)])

p += Grid(x0=-1, y0=-1, x1=1, y1=2)
print(p)
```

OR gate rotate by π:



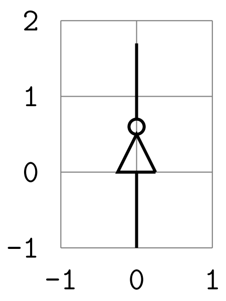
```
from math import pi
from latextool_basic import *
from latexcircuit import *
p = Plot()
g = OR_GATE(x=0, y=0, inputs=2, angle=pi)
p += str(g)

for x,y in g.inputs():
    p += Line(points=[(x + 1, y), (x, y)])

x,y = g.output()
p += Line(points=[(x, y),(x - 1, y)])

p += Grid(x0=-2, y0=-1, x1=1, y1=1)
print(p)
```

OR gate rotate by $3\pi/2$:



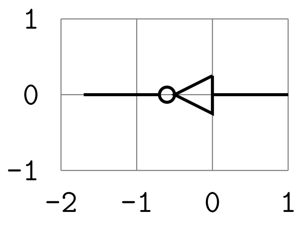
```
from math import pi
from latextool_basic import *
from latexcircuit import *
p = Plot()
g = OR_GATE(x=0, y=0, inputs=2, angle=3*pi/2)
p += str(g)

for x,y in g.inputs():
    p += Line(points=[(x, y + 1), (x, y)])

x,y = g.output()
p += Line(points=[(x, y),(x, y - 1)])

p += Grid(x0=-1, y0=-2, x1=1, y1=1)
print(p)
```

NOT gate rotate by $\pi/4$:



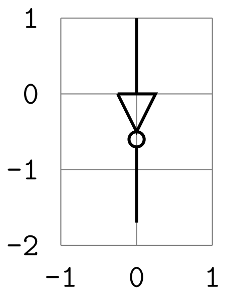
```
from math import pi
from latextool_basic import *
from latexcircuit import *
p = Plot()
g = NOT_GATE(x=0, y=0, inputs=2, angle=pi/2)
p += str(g)

x,y = g.input()
p += Line(points=[(x, y - 1), (x, y)])

x,y = g.output()
p += Line(points=[(x, y),(x, y + 1)])

p += Grid(x0=-1, y0=-1, x1=1, y1=2)
print(p)
```

NOT gate rotate by π:



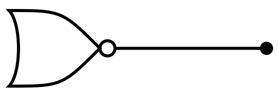
```
from math import pi
from latextool_basic import *
from latexcircuit import *
p = Plot()
g = NOT_GATE(x=0, y=0, inputs=2, angle=pi)
p += str(g)

x,y = g.input()
p += Line(points=[(x + 1, y), (x, y)])

x,y = g.output()
p += Line(points=[(x, y),(x - 1, y)])

p += Grid(x0=-2, y0=-1, x1=1, y1=1)
print(p)
```

NOT gate rotate by $3\pi/2$:

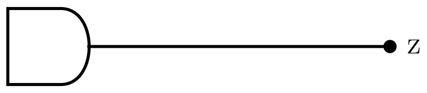


```
from math import pi
from latextool_basic import *
from latexcircuit import *
p = Plot()
g = NOT_GATE(x=0, y=0, inputs=2, angle=3*pi/2)
p += str(g)

x,y = g.input()
p += Line(points=[(x, y + 1), (x, y)])

x,y = g.output()
p += Line(points=[(x, y),(x, y - 1)])

p += Grid(x0=-1, y0=-2, x1=1, y1=1)
print(p)
```

Test OUTPUT POINT

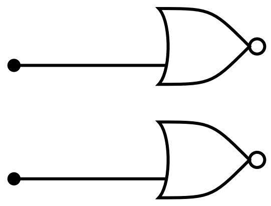


```
from latextool_basic import *
from latexcircuit import *
p = Plot()

NOR = NOR_GATE(x=0, y=3)
P = OUTPUT_POINT(gate=NOR)

p += str(NOR)
p += str(P)
print(p)
```



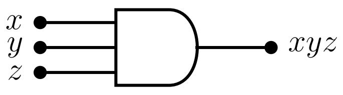
```
from latextool_basic import *
from latexcircuit import *
p = Plot()

AND = AND_GATE(x=0, y=3)
P = OUTPUT_POINT(gate=AND, output_length=4, label='z', anchor='west')

p += str(AND)
p += str(P)
print(p)
```

Test INPUT POINT



Example:

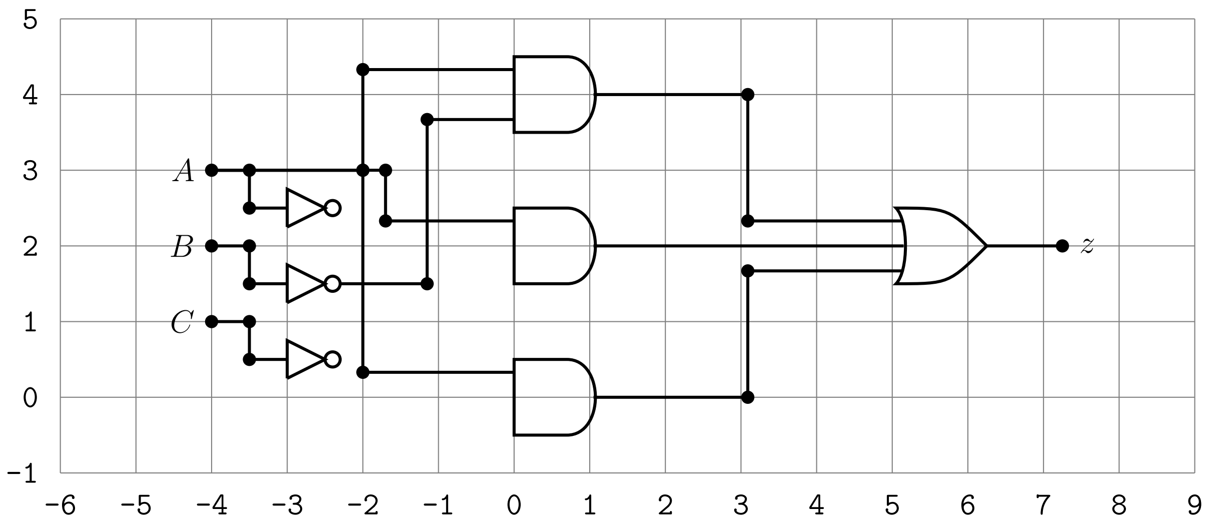
```
from latextool_basic import *
from latexcircuit import *
p = Plot()

AND = AND_GATE(x=0, y=3, inputs=3)
P0 = INPUT_POINT(gate=AND, input_index=2, label='$x$', input_length=1)
P1 = INPUT_POINT(gate=AND, input_index=1, label='$y$', input_length=1)
P2 = INPUT_POINT(gate=AND, input_index=0, label='$z$', input_length=1)
P3 = OUTPUT_POINT(gate=AND, label='$xyz$', anchor='west', output_length=1)

p += str(AND)
p += str(P0)
p += str(P1)
p += str(P2)
p += str(P3)
print(p)
```



Example:

```
from latextool_basic import *
from latexcircuit import *
p = Plot()

and0 = AND_GATE(x=0, y=0, h=1, inputs=3)

x, y = and0.inputs()[2]; x -= 1
X = POINT(x=x, y=y, label='$x$')

x, y = and0.inputs()[1]; x -= 1
Y = POINT(x=x, y=y, label='$y$')

x, y = and0.inputs()[0]; x -= 1
Z = POINT(x=x, y=y, label='$z$')

x, y = and0.output(); x += 1
XYZ = POINT(x=x, y=y, label='$xyz$', anchor='west')

p += str(and0)
p += str(X); p += str(Y); p += str(Z)
p += str(XYZ)

p += '%s' % OrthogonalPath([X.output(), and0.inputs()[2]])
p += '%s' % OrthogonalPath([Y.output(), and0.inputs()[1]])
p += '%s' % OrthogonalPath([Z.output(), and0.inputs()[0]])

p += '%s' % OrthogonalPath([XYZ.output(), and0.output()])

p += Grid(x0=-2, y0=-1, x1=4, y1=1)
print(p)
```

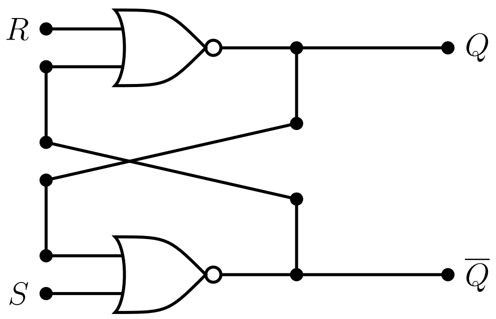
Example:



```
from latextool_basic import *
from latexcircuit import *
p = Plot()

and0 = AND_GATE(x=0,y=0,h=1, inputs=3)
and1 = AND_GATE(x=0,y=2,h=1, inputs=3)
and2 = AND_GATE(x=0,y=4,h=1, inputs=3)

x,y = and1.output(); x += 4
or0 = OR_GATE(x=x, y=y,h=1, inputs=3)

x, y = and1.inputs()[1]; x -= 4
A = POINT(x=x, y=y + 1, label='$A$')
B = POINT(x=x, y=y + 0, label='$B$')
C = POINT(x=x, y=y + -1, label='$C$')
notA = NOT_GATE(x=A.x()+1, y=A.y()-0.5)
notB = NOT_GATE(x=B.x()+1, y=B.y()-0.5)
notC = NOT_GATE(x=C.x()+1, y=C.y()-0.5)

x0,y0 = or0.output()
Z = POINT(x=x0+1, y=y0, label='$z$', anchor='west')

p += str(A); p += str(B); p += str(C)
p += str(notA); p += str(notB); p += str(notC)
p += str(and0); p += str(and1); p += str(and2)
p += str(or0)

p += str(Z)

OP = OrthogonalPath
# Join A to notA, ...
p += '%s' % OP([A.output(), notA.input()])
p += '%s' % OP([B.output(), notB.input()])
p += '%s' % OP([C.output(), notC.input()])

# Join inputs to AND gates
p += '%s' % OP([A.output(), and0.inputs()[2]])
p += '%s' % OP([A.output(), and1.inputs()[2]], shifts=[0.3])
p += '%s' % OP([A.output(), and2.inputs()[2]])
p += '%s' % OP([notB.output(), and2.inputs()[0]])

# Join AND gates to OR
p += '%s' % OP([and0.output(), or0.inputs()[0]])
p += '%s' % OP([and1.output(), or0.inputs()[1]])
p += '%s' % OP([and2.output(), or0.inputs()[2]])

# Join OR to z
p += '%s' % OP([or0.output(), Z.input()])

p += Grid(x0=-6, y0=-1, x1=9, y1=5)
print(p)
```

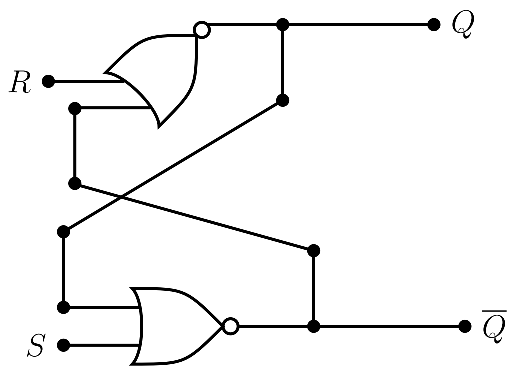
47.13 Example: SR latch

SR latch:



```
from latextool_basic import *
from latexcircuit import *
p = Plot()

NOR1 = NOR_GATE(x=0, y=3)
R = INPUT_POINT(gate=NOR1, input_index=1, label='$R$', input_length=1)
R0 = INPUT_POINT(gate=NOR1, input_index=0, input_length=1)
OUT1 = OUTPUT_POINT(gate=NOR1, output_length=1)
x,y = OUT1.output()
OUT11 = POINT(x=x+2, y=y, label='$Q$', anchor='west')

NOR2 = NOR_GATE(x=0, y=0)
S = INPUT_POINT(gate=NOR2, input_index=0, label='$S$', input_length=1)
S0 = INPUT_POINT(gate=NOR2, input_index=1, input_length=1)
OUT2 = OUTPUT_POINT(gate=NOR2, output_length=1)
x,y = OUT2.output()
OUT22 = POINT(x=x+2, y=y, label='$\overline{Q}$', anchor='west')

OP = OrthogonalPath

p += str(NOR1)
p += str(R)
p += str(R0)
p += str(OUT1); p += str(OUT11)

p += str(NOR2)
p += str(S)
p += str(S0)
p += str(OUT2); p += str(OUT22)

p += str(OP([OUT1.output(), OUT11.input()]))
p += str(OP([OUT2.output(), OUT22.input()]))

# Cross
x0,y0 = R0.input()
x1,y1 = OUT2.input()
p += Line(points=[(x0,y0),(x0,y0-1),(x1,y1+1),(x1,y1)])
p += str(POINT(x=x0,y=y0-1))
p += str(POINT(x=x1,y=y1+1))

x0,y0 = S0.input()
x1,y1 = OUT1.input()
p += Line(points=[(x0,y0),(x0,y0+1),(x1,y1-1),(x1,y1)])
p += str(POINT(x=x0,y=y0+1))
p += str(POINT(x=x1,y=y1-1))

print(p)
```

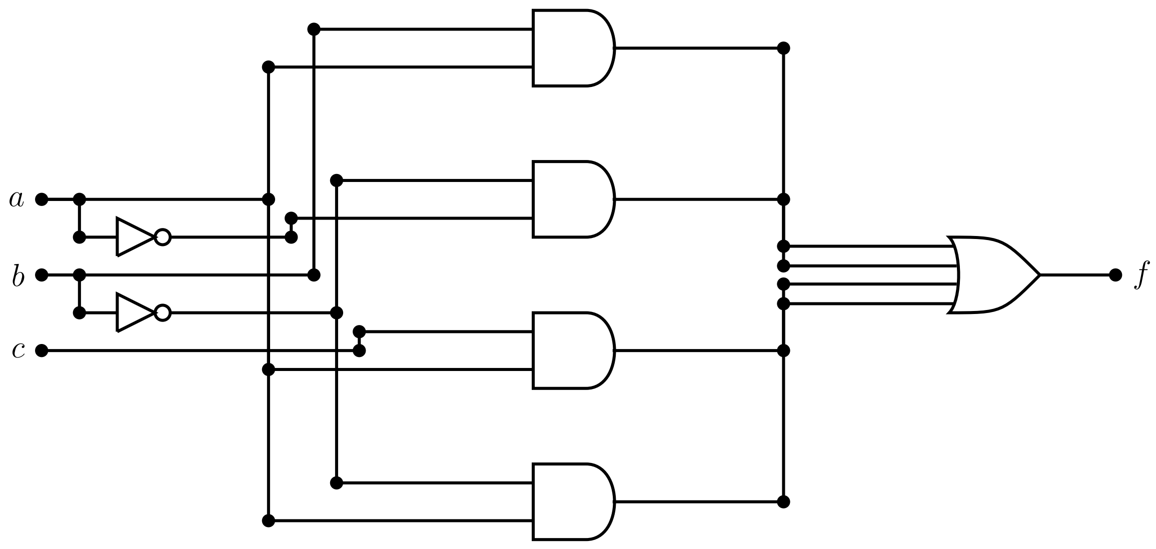
Test angle = 3.14159/4:



```
from latextool_basic import *
from latexcircuit import *
p = Plot()

NOR1 = NOR_GATE(x=0, y=3, angle=3.14159/4)
R = INPUT_POINT(gate=NOR1, input_index=1, label='$R$', input_length=1)
R0 = INPUT_POINT(gate=NOR1, input_index=0, input_length=1)
OUT1 = OUTPUT_POINT(gate=NOR1, output_length=1)
x,y = OUT1.output()
OUT11 = POINT(x=x+2, y=y, label='$Q$', anchor='west')

NOR2 = NOR_GATE(x=0, y=0)
S = INPUT_POINT(gate=NOR2, input_index=0, label='$S$', input_length=1)
S0 = INPUT_POINT(gate=NOR2, input_index=1, input_length=1)
OUT2 = OUTPUT_POINT(gate=NOR2, output_length=1)
x,y = OUT2.output()
OUT22 = POINT(x=x+2, y=y, label='$\overline{Q}$', anchor='west')

OP = OrthogonalPath

p += str(NOR1)
p += str(R)
p += str(R0)
p += str(OUT1); p += str(OUT11)

p += str(NOR2)
p += str(S)
p += str(S0)
p += str(OUT2); p += str(OUT22)

p += str(OP([OUT1.output(), OUT11.input()]))
p += str(OP([OUT2.output(), OUT22.input()]))

# Cross
x0,y0 = R0.input()
x1,y1 = OUT2.input()
p += Line(points=[(x0,y0),(x0,y0-1),(x1,y1+1),(x1,y1)])
p += str(POINT(x=x0,y=y0-1))
p += str(POINT(x=x1,y=y1+1))

x0,y0 = S0.input()
x1,y1 = OUT1.input()
p += Line(points=[(x0,y0),(x0,y0+1),(x1,y1-1),(x1,y1)])
p += str(POINT(x=x0,y=y0+1))
p += str(POINT(x=x1,y=y1-1))

print(p)
```

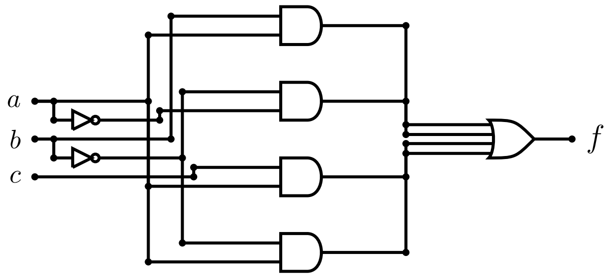
47.14 layout function

SOP example:



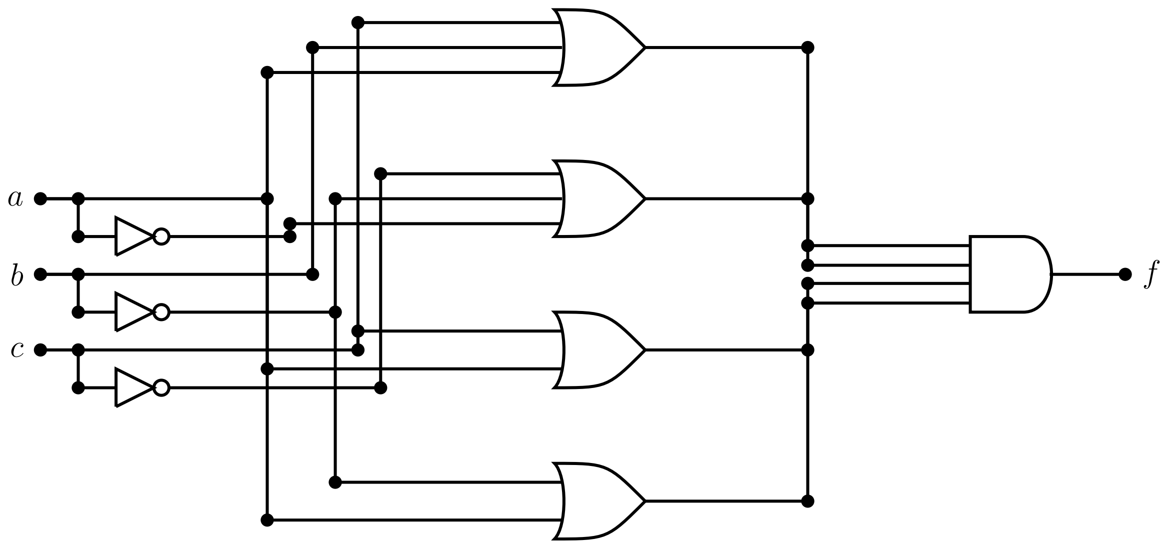
```
from latextool_basic import *
from latexcircuit import *
p = Plot()
expr = [["a", "b"], ["a'", "b'"], ["a", "c"], ["a", "b'"]]
layout(p, expr, AND_GATE, OR_GATE)
print(p)
```

layout with scale and font for label:



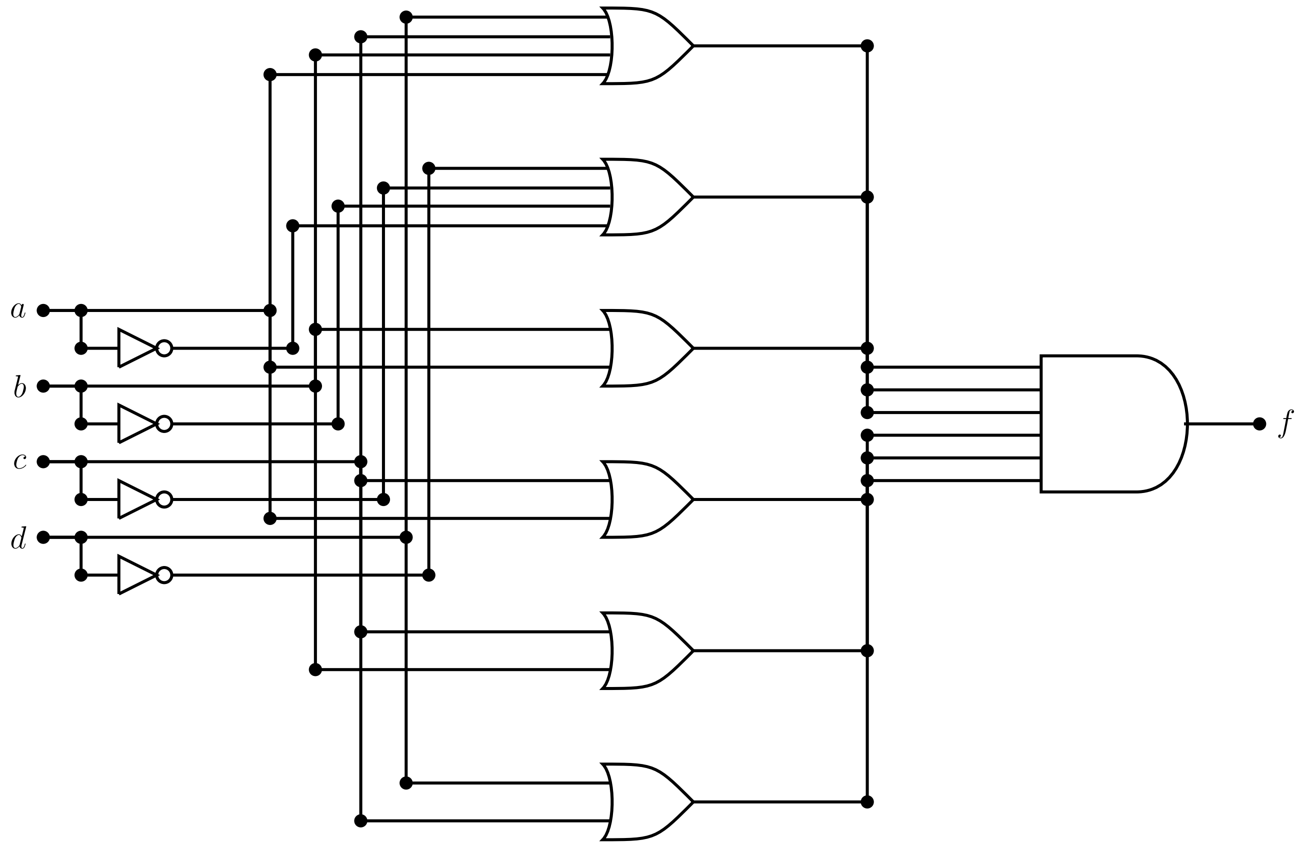
```
from latextool_basic import *
from latexcircuit import *
p = Plot(scale=0.5)
expr = [["a", "b"], ["a'", "b'"], ["a", "c"], ["a", "b'"]]
layout(p, expr, AND_GATE, OR_GATE, font='footnotesize')
print(p)
```

POS example:



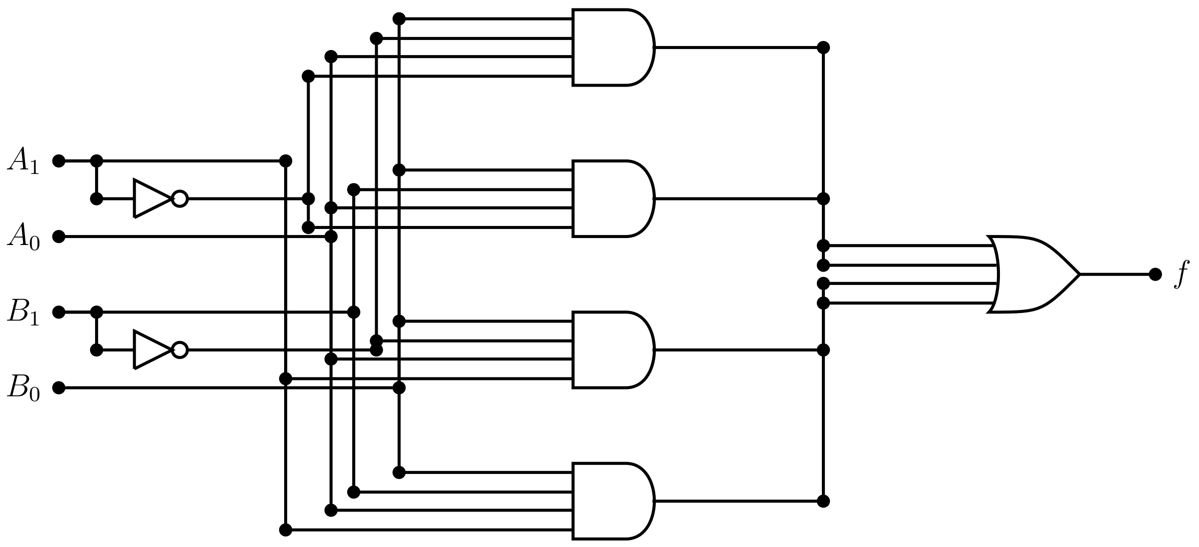
```
from latextool_basic import *
from latexcircuit import *
p = Plot()
expr = [["a", "b", "c"], ["a'", "b'", "c'"], ["a", "c"], ["a", "b'"]]
layout(p, expr, OR_GATE, AND_GATE)
print(p)
```

POS example:



```
from latextool_basic import *
from latexcircuit import *
p = Plot()
expr = [["a", "b", "c", "d"], ["a'", "b'", "c'", "d'"],
        ["a", "b"], ["a", "c"], ["b","c"], ["c","d"]]
layout(p, expr, OR_GATE, AND_GATE)
print(p)
```

POS example:



```
from latextool_basic import *
from latexcircuit import *
p = Plot()
expr = [["A_1'", "A_0", "B_1'", "B_0"],
        ["A_1'", "A_0", "B_1", "B_0"],
        ["A_1", "A_0", "B_1'", "B_0"],
        ["A_1", "A_0", "B_1", "B_0"],
        ]
layout(p, expr, AND_GATE, OR_GATE)
print(p)
```

SOP example: PYTHON ERROR. See source, stderr, stdout below

```
from latextool_basic import console
from latexcircuit import SOP2
s = SOP("a'bcd + a'bc'd + abc'd + abcd")
print(s)
```

```
```

```
Traceback (most recent call last):
  File "myviytlb.tmp.py", line 3, in <module>
    s = SOP("a'bcd + a'bc'd + abc'd + abcd")
NameError: name 'SOP' is not defined
```

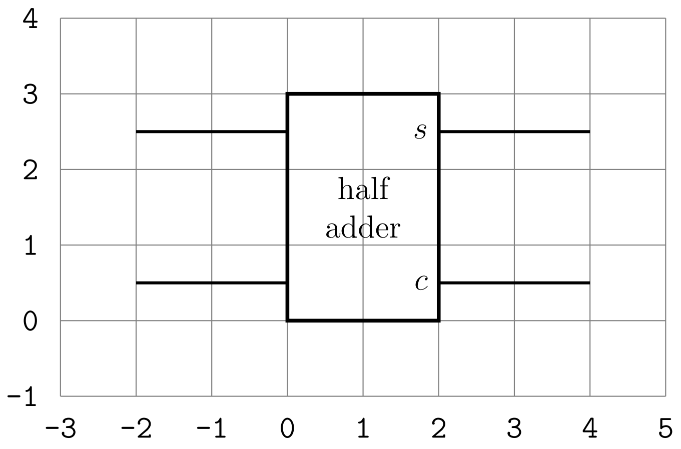
```
from latextool_basic import console
from latexcircuit import SOP2
s = SOP("a'bcd + a'bc'd + abc'd + abcd")
print(s)
```

47.15 Logic blocks

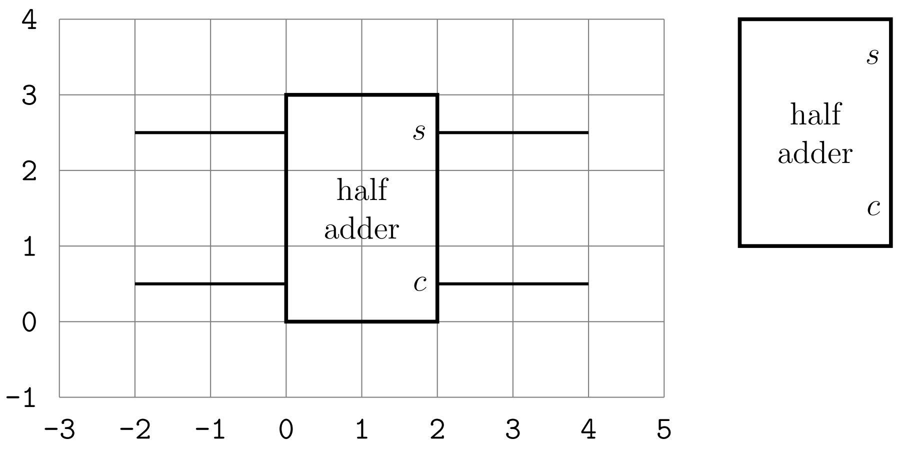
`points` argument is a dictionary.

`points[k]` is `(label, coordinates)`. `label` is used to label the point in the block. `coordinate` is the coordinates of the point related to the block.

You can get the coordinates of a point by calling the `block.point(k)`.

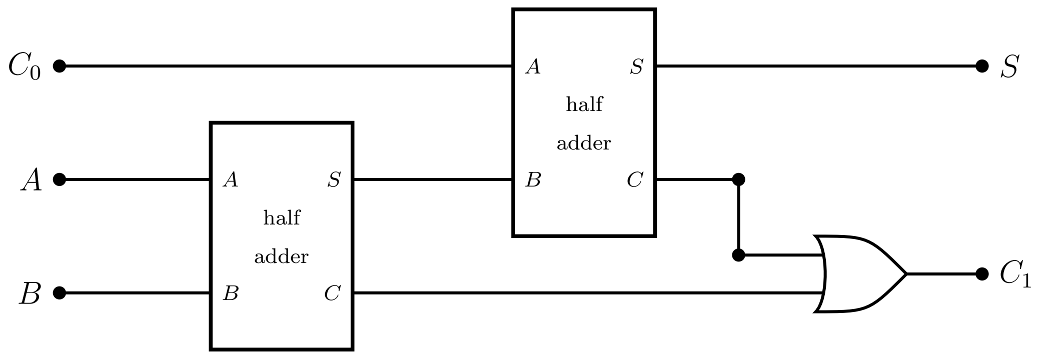


```
from latextool_basic import *
from latexcircuit import *
p = Plot()
g = LOGIC_BLOCK(x=0, y=0, w=2, h=3, s='half adder',
                points={'s':('$s$', (2,2.5)),
                        'c':('$c$', (2,0.5)),
                        'x':('', (0,2.5)),
                        'y':('', (0,0.5))})
p += str(g)

x,y = g.point('x'); P0 = POINT(x=x-2, y=y, label='$x$', anchor='east')
p += str(OrthogonalPath([P0.output(), (x,y)]))

x,y = g.point('y'); P1 = POINT(x=x-2, y=y, label='$y$', anchor='east')
p += str(OrthogonalPath([P1.output(), (x,y)]))

x,y = g.point('s'); P2 = POINT(x=x+2, y=y, label='$s$', anchor='east')
p += str(OrthogonalPath([P2.output(), (x,y)]))

x,y = g.point('c'); P2 = POINT(x=x+2, y=y, label='$c$', anchor='east')
p += str(OrthogonalPath([P2.output(), (x,y)]))

p += Grid(-3, -1, 5, 4)
print(p)
```

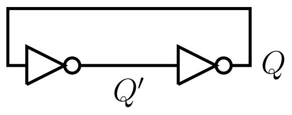


```
from latextool_basic import *
from latexcircuit import *
p = Plot()
g = LOGIC_BLOCK(x=0, y=0, w=2, h=3, s='half adder',
                points={'s':('$s$', (2,2.5)),
                        'c':('$c$', (2,0.5)),
                        'x':('', (0,2.5)),
                        'y':('', (0,0.5))})
p += str(g)

x,y = g.point('x'); P0 = POINT(x=x-2, y=y, label='$x$', anchor='east')
p += str(OrthogonalPath([P0.output(), (x,y)]))

x,y = g.point('y'); P1 = POINT(x=x-2, y=y, label='$y$', anchor='east')
p += str(OrthogonalPath([P1.output(), (x,y)]))

x,y = g.point('s'); P2 = POINT(x=x+2, y=y, label='$s$', anchor='east')
p += str(OrthogonalPath([P2.output(), (x,y)]))

x,y = g.point('c'); P2 = POINT(x=x+2, y=y, label='$c$', anchor='east')
p += str(OrthogonalPath([P2.output(), (x,y)]))

p += '%s' % LOGIC_BLOCK(x=6, y=1, h=3, s='half adder',
                        points={'s':('$s$', (2,2.5)),
                                'c':('$c$', (2,0.5)),
                                'x':('', (0,2.5)),
                                'y':('', (0,0.5))})
p += Grid(-3, -1, 5, 4)
print(p)
```

47.16 Logic blocks: full adder



47.17 Two NOTs



```
from latextool_basic import *
from latexcircuit import *
p = Plot()

g0 = NOT_GATE(x=0, y=0)
x,y = g0.output()
g1 = NOT_GATE(x=2, y=y)

p += str(g0)
p += str(g1)

p += Line(points=[g0.output(), g1.input()], label="$Q'$", anchor='below')

x0, y0 = g1.output()
x5, y5 = g0.input()

dx = 0.25
dy = 0.75

x1, y1 = x0+dx, y0
x2, y2 = x1, y1+dy
x4, y4 = x5-dx, y5
x3, y3 = x4, y2
p += Line(points=[(x0, y0),
                  (x1, y1),
                  (x2, y2),
                  (x3, y3),
                  (x4, y4),
                  (x5, y5)])
X = POINT(x=x1, y=y1, r=0, label='$Q$', anchor='west')
p += str(X)
print(p)
```

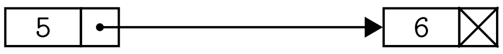
debug: chap-
structures.tex

48 Data structures

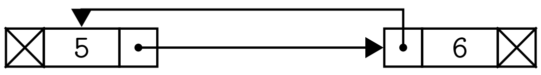
debug:
singlylinkedli

48.1 Singly Linked List

```
from latextool_basic import *
p = Plot()

SLLNode = SinglyLinkedListNode

n5 = SLLNode(x=0, y=0, label=5)
p += n5

n6 = SLLNode(x=5, y=0, label=6, next=None)
p += n6

p += Pointer(points = [n5[1].center(), n6[0].left()])

print(p)
```



```
from latextool_basic import *
p = Plot()

DLLNode = DoublyLinkedListNode

n5 = DLLNode(x=0, y=0, prev=None, label=5)
p += n5

n6 = DLLNode(x=5, y=0, label=6, next=None)
p += n6

p += Pointer(points = [n5[2].center(), n6[0].left()])

p0 = n6[0].center()
p1 = n5[1].top()
p2 = p0[0], p0[1] + 0.5
p3 = p1[0], p2[1]
p += Pointer(points = [p0, p2, p3, p1])
print(p)
```



debug:
doublylinkedl

48.2 Doubly linked list

```
from latextool_basic import *
class DLNodeRect:
    L = 0.05 # linewidth
    def __init__(self, x0=0, y0=0, w=0, h=0, label='',
                 prev=None, next=None):
        L = DLNodeRect.L
        self.x0, self.y0 = x0, y0
        self.w, self.h = w, h
        self.label = label
        self.prev, self.next = prev, next
        c = RectContainer(x0=x0, y0=y0)
        c += Rect2(x0=0, y0=0, x1=w, y1=h, linewidth=L, label='')
        c += Rect2(x0=0, y0=0, x1=w, y1=h, linewidth=L, label=label)
        c += Rect2(x0=0, y0=0, x1=w, y1=h, linewidth=L, label='')
        self.c = c
    def next_rect(self): return self.c[2]
    def key_rect(self): return self.c[1]
    def prev_rect(self): return self.c[0]
    def __str__(self):
        L = DLNode.L
        s = str(self.c)
        if self.next != None:
            a = self.next_rect().center()
            b = self.next.prev_rect().left()
            s += line(points=[a, b], linewidth=L,
                      endstyle='->', startstyle='dot', arrowstyle='triangle
')
        else: # self.next is None
            r = self.next_rect()
            s += line(points=[r.topleft(), r.bottomright()], linewidth=L)
            s += line(points=[r.topright(), r.bottomleft()], linewidth=L)
        if self.prev != None:
            a = self.prev_rect().center()
            b = self.prev.key_rect().bottom()
            c = a[0], a[1] - 1
            d = b[0], c[1]
            s += line(points=[a, c, d, b], linewidth=L,
                      endstyle='->', startstyle='dot', arrowstyle='triangle
')
        else: # self.prev is None
            r = self.prev_rect()
            s += line(points=[r.topleft(), r.bottomright()], linewidth=L)
            s += line(points=[r.topright(), r.bottomleft()], linewidth=L)
        return s

class DLNode:
    def __init__(self, key, prev=None, next=None):
        self.key = key
        self.prev, self.next = prev, next
    def __str__(self):
        return "<DLNode: %s, %s, %s>" % (self.prev, self.key, self.next)

c = DLNode(x0=1, y0=1, w=1, h=1, label=r'{\texttt 0}')
d = DLNode(x0=5, y0=1, w=1, h=1, label=r'{\texttt 1}')
e = DLNode(x0=9, y0=1, w=1, h=1, label=r'{\texttt 2}')
c.next = d; d.prev = c; d.next = e; e.prev = d

p = Plot()
p += str(c); p += str(d); p += str(e)
print(p)
```

PYTHON ERROR. See source, stderr, stdout below

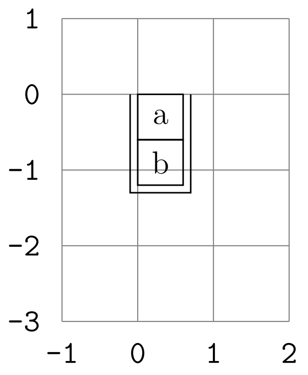
```
from latextool_basic import *
class DLNodeRect:
    L = 0.05 # linewidth
    def __init__(self, x0=0, y0=0, w=0, h=0, label='',
                 prev=None, next=None):
        L = DLNodeRect.L
        self.x0, self.y0 = x0, y0
        self.w, self.h = w, h
        self.label = label
        self.prev, self.next = prev, next
        c = RectContainer(x0=x0, y0=y0)
        c += Rect2(x0=0, y0=0, x1=w, y1=h, linewidth=L, label='')
        c += Rect2(x0=0, y0=0, x1=w, y1=h, linewidth=L, label=label)
        c += Rect2(x0=0, y0=0, x1=w, y1=h, linewidth=L, label='')
        self.c = c
    def next_rect(self): return self.c[2]
    def key_rect(self): return self.c[1]
    def prev_rect(self): return self.c[0]
    def __str__(self):
        L = DLNode.L
        s = str(self.c)
        if self.next != None:
            a = self.next_rect().center()
            b = self.next.prev_rect().left()
            s += line(points=[a, b], linewidth=L,
                      endstyle='->', startstyle='dot', arrowstyle='triangle
')
        else: # self.next is None
            r = self.next_rect()
            s += line(points=[r.topleft(), r.bottomright()], linewidth=L)
            s += line(points=[r.topright(), r.bottomleft()], linewidth=L)
        if self.prev != None:
            a = self.prev_rect().center()
            b = self.prev.key_rect().bottom()
            c = a[0], a[1] - 1
            d = b[0], c[1]
            s += line(points=[a, c, d, b], linewidth=L,
                      endstyle='->', startstyle='dot', arrowstyle='triangle
')
        else: # self.prev is None
            r = self.prev_rect()
            s += line(points=[r.topleft(), r.bottomright()], linewidth=L)
            s += line(points=[r.topright(), r.bottomleft()], linewidth=L)
        return s

class DLNode:
    def __init__(self, key, prev=None, next=None):
        self.key = key
        self.prev, self.next = prev, next
    def __str__(self):
        return "<DLNode: %s, %s, %s>" % (self.prev, self.key, self.next)

c = DLNode(x0=1, y0=1, w=1, h=1, label=r'{\texttt 0}')
d = DLNode(x0=5, y0=1, w=1, h=1, label=r'{\texttt 1}')
e = DLNode(x0=9, y0=1, w=1, h=1, label=r'{\texttt 2}')
c.next = d; d.prev = c; d.next = e; e.prev = d

p = Plot()
p += str(c); p += str(d); p += str(e)
print(p)
```

```
Traceback (most recent call last):
  File "arwyorcf.tmp.py", line 51, in <module>
    c = DLNode(x0=1, y0=1, w=1, h=1, label=r'{\texttt 0}')
TypeError: __init__() got an unexpected keyword argument 'x0'
```

debug: stack.

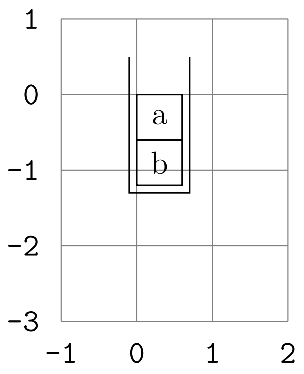
48.3 Stack

```
from latextool_basic import *
p = Plot()
drawstack(p,
  xs=['a', 'b'],
  )
p += Grid(-1, -3, 2, 1)
print(p)
```



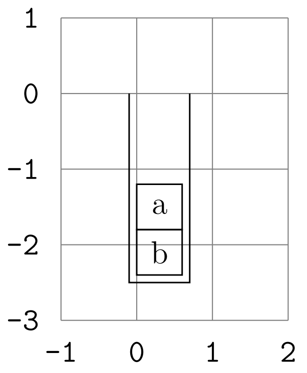
Extra height for container:

```
from latextool_basic import *
p = Plot()
drawstack(p,
  xs=['a', 'b'],
  extra_h=0.5
  )
p += Grid(-1, -3, 2, 1)
print(p)
```



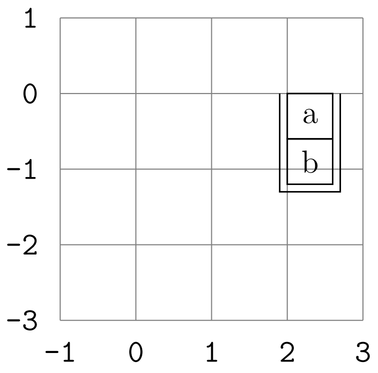
Minimum number of values:

```
from latextool_basic import *
p = Plot()
drawstack(p,
  xs=['a', 'b'],
  min_size=4
  )
p += Grid(-1, -3, 2, 1)
print(p)
```



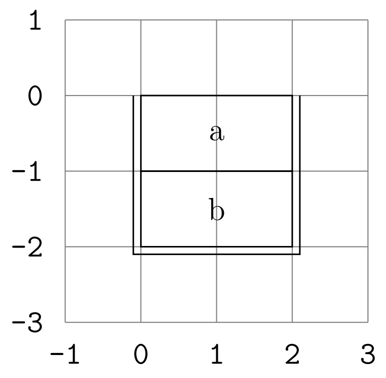
Position:

```
from latextool_basic import *
p = Plot()
drawstack(p,
  x=2, y=0,
  xs=['a', 'b'],
  )
p += Grid(-1, -3, 3, 1)
print(p)
```



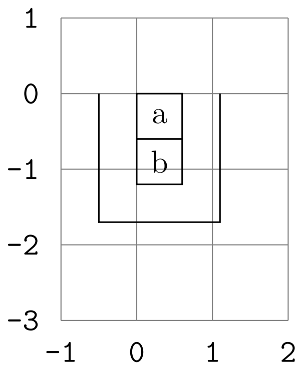
Width and height:

```
from latextool_basic import *
p = Plot()
drawstack(p,
  w=2, h=1,
  xs=['a', 'b'],
  )
p += Grid(-1, -3, 3, 1)
print(p)
```



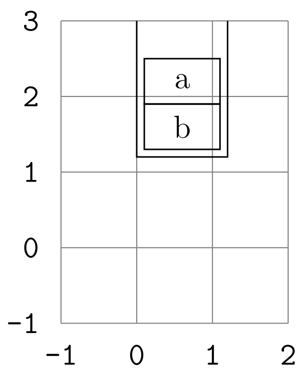
Distance of walls to values:

```
from latextool_basic import *
p = Plot()
drawstack(p,
  xs=['a', 'b'],
  sep=0.5,
  )
p += Grid(-1, -3, 2, 1)
print(p)
```



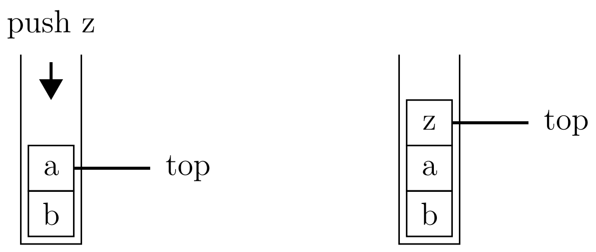
To make bottom right of container at (0,0):

```
from latextool_basic import *
p = Plot()

sep = 0.1
w = 1; h = 0.6
x = sep; y = sep + 4*h
drawstack(p,
  x=x, y=y, w=w, h=h, extra_h=0.5,
  xs=['a', 'b'],
  sep=sep,
  )
p += Grid(-1, -1, 2, 3)
print(p)
```



Return value is a RectContainer:

```
from latextool_basic import *
p = Plot()

r0 = drawstack(p,
     x=0, y=0,
     xs=['a', 'b'],
     min_size=4
     )
x0,y0 = r0[2].right()
x1,y1 = x0 + 1, y0
p += Line(points=[(x0,y0),(x1,y1)])
p += Rect(x0=x1+0.5,y0=y0,x1=x1+0.5,y1=y0,label='top', linewidth=0)

x0,y0 = r0[1].top(); y0 + 0.5
x1,y1 = x0,y0+1
p += Rect(x0=x1, y0=y1, x1=x1,y1=y1,label='push z', linewidth=0)

y1 -= 0.5
p += Line(points=[(x1,y1),(x0,y0)], endstyle='>')

r1 = drawstack(p,
     x=5, y=0,
     xs=['z', 'a', 'b'],
     min_size=4
     )
x0,y0 = r1[1].right()
x1,y1 = x0 + 1, y0
p += Line(points=[(x0,y0),(x1,y1)])
p += Rect(x0=x1+0.5,y0=y0,x1=x1+0.5,y1=y0,label='top', linewidth=0)

print(p)
```

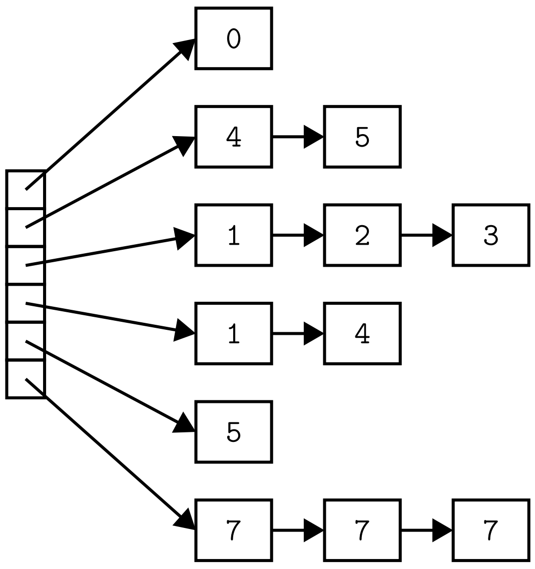


debug: adjlis

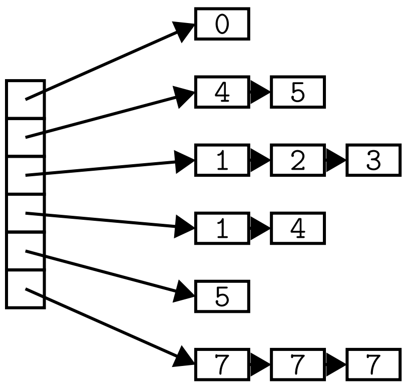
48.4 Adjacency list

Even number of linked lists:

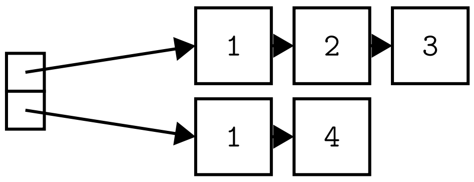
```
from latextool_basic import *
p = Plot()

r, crosses, nodes, lines = adjlist(xs=[[0],[4,5],[1,2,3],[1,4], [5],[7,7,7]
],
                                   nodevspace=0.5, nodehspace=0.7,
                                   nodewidth=1, nodeheight=0.8)

p += r

for k,v in crosses.items():
    for _ in v: p += _

for k,v in nodes.items():
    for _ in v: p += _

for k,v in lines.items():
    for _ in v: p += _

print(p)
```



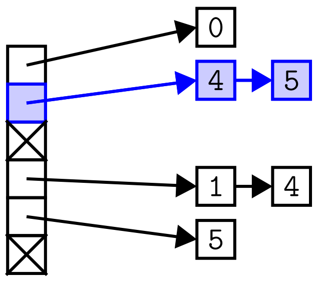
```
from latextool_basic import *
p = Plot()

r, crosses, nodes, lines = adjlist(xs=[[0],[4,5],[1,2,3],[1,4], [5],[7,7,7]
],
                                   nodevspace=0.5, nodehspace=0.3,
                                   nodewidth=0.7, nodeheight=0.4)

p += r

for k,v in crosses.items():
    for _ in v: p += _

for k,v in nodes.items():
    for _ in v: p += _

for k,v in lines.items():
    for _ in v: p += _

print(p)
```



```
from latextool_basic import *
p = Plot()

r, crosses, nodes, lines = adjlist(xs=[[1,2,3],[1,4],],
                                   nodevspace=0.2, nodehspace=0.3,
                                   nodewidth=1, nodeheight=1)

p += r

for k,v in crosses.items():
    for _ in v: p += _

for k,v in nodes.items():
    for _ in v: p += _

for k,v in lines.items():
    for _ in v: p += _

print(p)
```

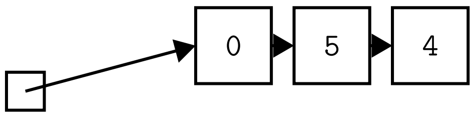


```
from latextool_basic import *
p = Plot()

r, crosses, nodes, lines = adjlist(xs=[[0],[4,5],[], [1,4], [5],[]])
p += r

r0 = r[1] # redraw r[1] with 'blue!20'
r0.background = 'blue!20'
r0.linecolor = 'blue'
p += r0

for k,v in crosses.items():
    for _ in v: p += _

for k,v in nodes.items():
    for _ in v: p += _

# redraw nodes[1]
for _ in nodes[1]:
    _[0].background = 'blue!20'
    _[0].linecolor = 'blue'
    p += _[0]

for k,v in lines.items():
    for _ in v: p += _

# redraw lines[1]
for _ in lines[1]:
    _.linecolor = 'blue'
    p += _

print(p)
```

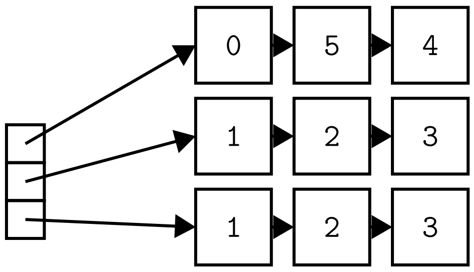


Test odd number of pointers

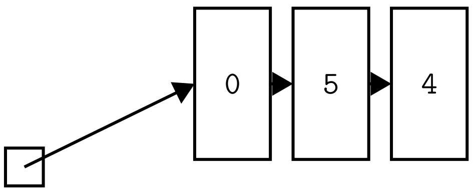
```
from latextool_basic import *
p = Plot()

r, crosses, nodes, lines = adjlist(xs=[[0,5,4]],
                                   nodevspace=0.2, nodehspace=0.3,
                                   nodewidth=1, nodeheight=1)

p += r
for k,v in crosses.items():
    for _ in v: p += _
for k,v in nodes.items():
    for _ in v: p += _
for k,v in lines.items():
    for _ in v: p += _

print(p)
```



```
from latextool_basic import *
p = Plot()

r, crosses, nodes, lines = adjlist(xs=[[0,5,4],[1,2,3],[1,2,3]],
                                   nodevspace=0.2, nodehspace=0.3,
                                   nodewidth=1, nodeheight=1)

p += r
for k,v in crosses.items():
    for _ in v: p += _
for k,v in nodes.items():
    for _ in v: p += _
for k,v in lines.items():
    for _ in v: p += _

print(p)
```

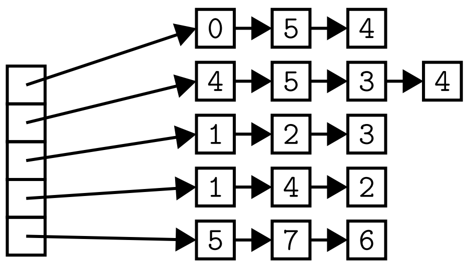


```
from latextool_basic import *
p = Plot()

r, crosses, nodes, lines = adjlist(xs=[[0,5,4]],
                                   nodevspace=0.2, nodehspace=0.3,
                                   nodewidth=1, nodeheight=2)

p += r
for k,v in crosses.items():
    for _ in v: p += _
for k,v in nodes.items():
    for _ in v: p += _
for k,v in lines.items():
    for _ in v: p += _

print(p)
```



```
from latextool_basic import *
p = Plot()

r, crosses, nodes, lines = adjlist(xs=[[0,5,4]],
                                   nodevspace=0.2, nodehspace=0.3,
                                   nodewidth=1, nodeheight=1)
)
p += r
for k,v in crosses.items():
    for _ in v: p += _
for k,v in nodes.items():
    for _ in v: p += _
for k,v in lines.items():
    for _ in v: p += _

print(p)
```

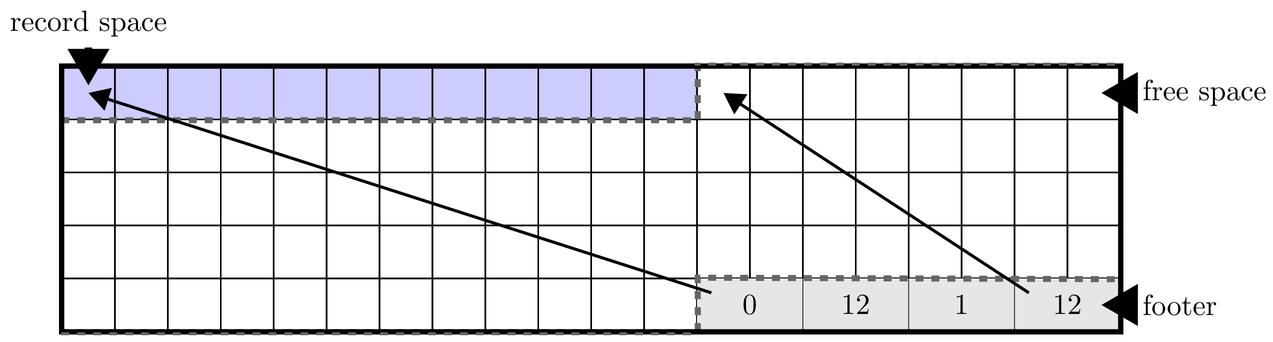
```
from latextool_basic import *
p = Plot()

r, crosses, nodes, lines = adjlist(xs=[[0,5,4],[4,5,3,4],[1, 2, 3], [1,4,2]
, [5,7,6]])
p += r
for k,v in crosses.items():
    for _ in v: p += _
for k,v in nodes.items():
    for _ in v: p += _
for k,v in lines.items():
    for _ in v: p += _

print(p)
```



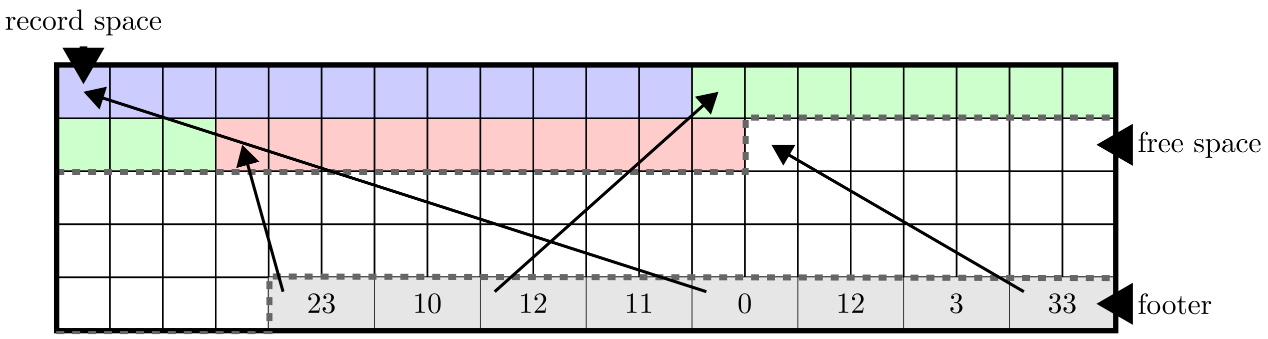
debug: heapfi

48.5 Heap file

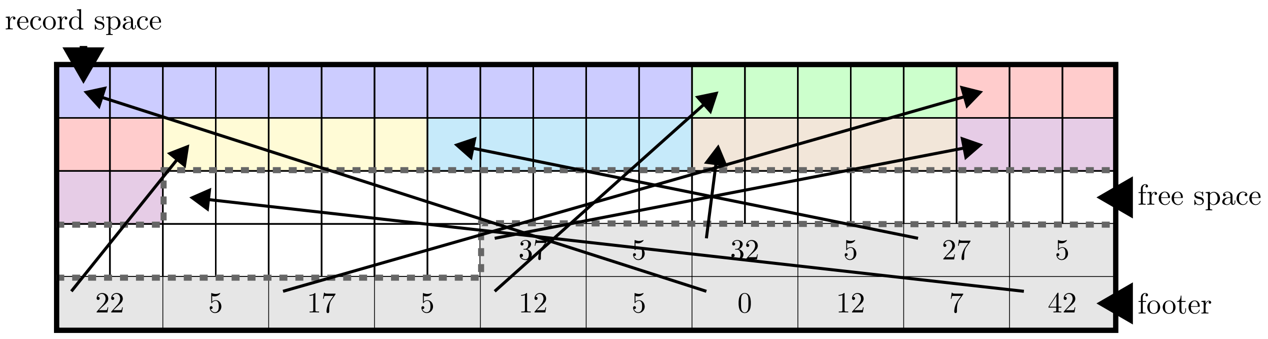
```
from latextool_basic import *
p = Plot()

nrows = 5
ncols = 20
slots = [(0,12)]

heapfilepage(p, nrows, ncols, slots)
print(p)
```



```
from latextool_basic import *

p = Plot()

nrows = 5
ncols = 20
slots = [(0, 12), (12, 11), (23, 10)]

heapfilepage(p, nrows, ncols, slots)
print(p)
```

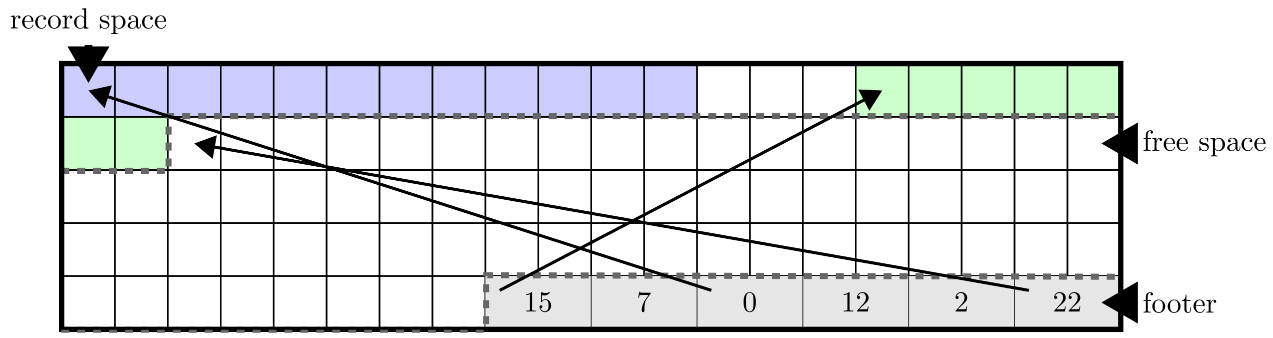


```
from latextool_basic import *

p = Plot()

nrows = 5
ncols = 20
slots = [(0,12), (12,5), (17,5), (22,5), (27,5), (32,5), (37,5)]

heapfilepage(p, nrows, ncols, slots)
print(p)
```



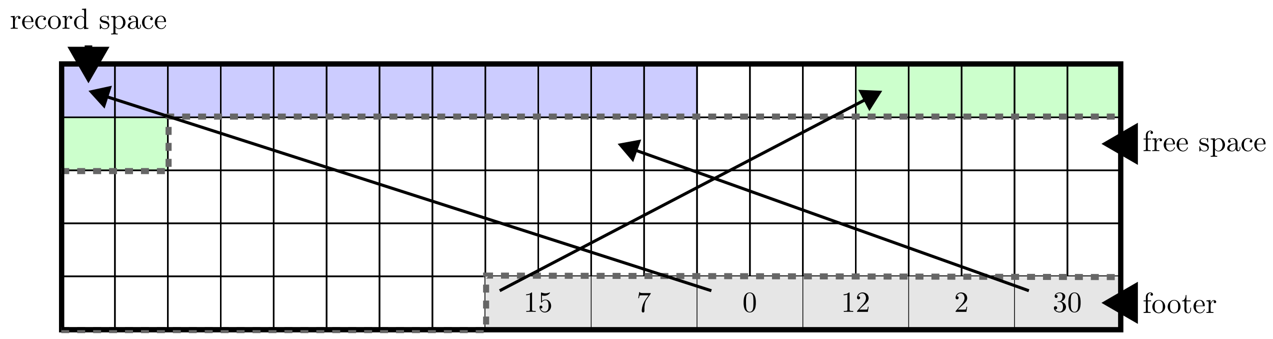
Test with holes:

```
from latextool_basic import *
p = Plot()

nrows = 5
ncols = 20
slots = [(0,12), (15, 7)]

heapfilepage(p, nrows, ncols, slots)
print(p)
```



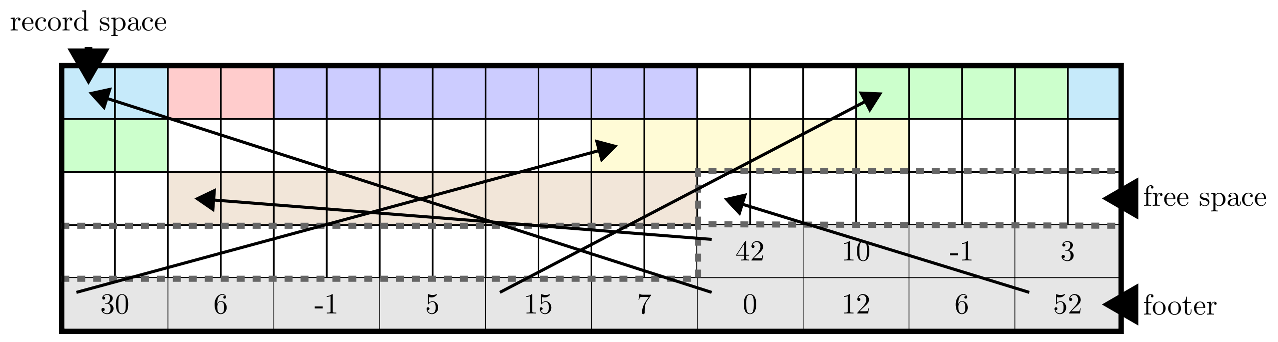
`free_offset`:

```
from latextool_basic import *
p = Plot()

nrows = 5
ncols = 20
slots = [(0,12), (15, 7)]
free_offset = 30

heapfilepage(p, nrows, ncols, slots, free_offset)
print(p)
```



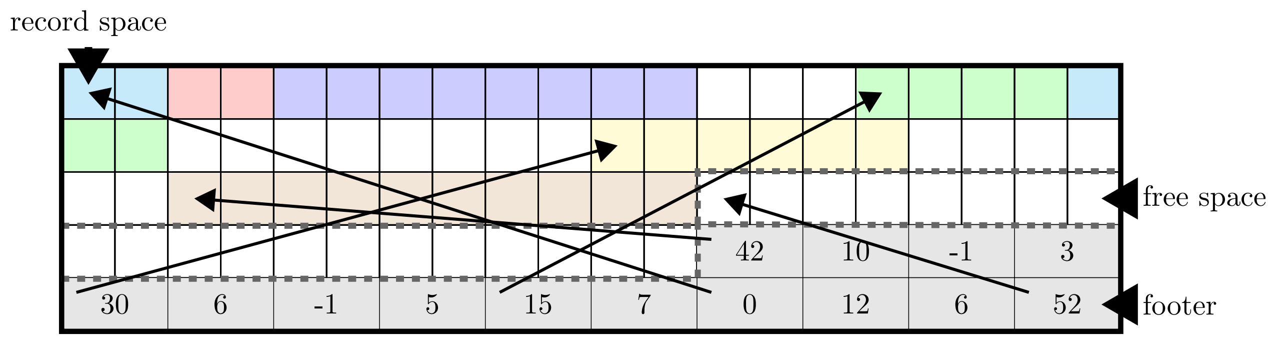
deleted records:

```
from latextool_basic import *
p = Plot()

nrows = 5
ncols = 20
slots = [(0,12), (15, 7), (-1, 5), (30, 6), (-1, 3), (42, 10)]

heapfilepage(p, nrows, ncols, slots)
print(p)
```



deleted records:

```
from latextool_basic import *
p = Plot()

nrows = 5
ncols = 20
slots = [(0,12), (15, 7), (-1, 5), (30, 6), (-1, 3), (42, 10)]

heapfilepage(p, nrows, ncols, slots)
print(p)
```

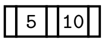


debug: bptre

49 B+ tree

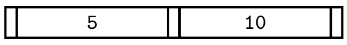
49.1 Node

```
from latextool_basic import *
p = Plot()

widths= [0.15, 0.4]; height = 0.4

def f(M):
    return [r'{{\scriptsize\texttt{%s}}}' % _ for _ in M]

M0 = f([5, 10])

Y = 0
N0 = bpt_node(x=4.5, y=Y, M=M0, widths=widths, height=height)

p += N0
print(p)
```



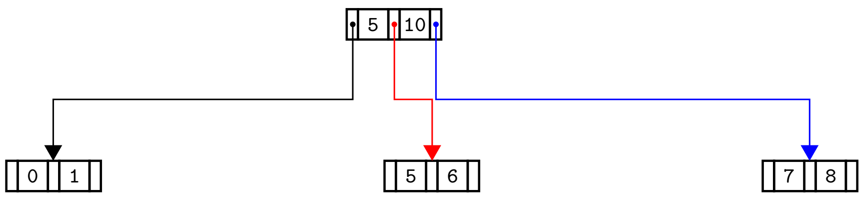
Key width:

```
from latextool_basic import *
p = Plot()

widths= [0.15, 2]; height = 0.4

def f(M):
    return [r'{{\scriptsize\texttt{%s}}}' % _ for _ in M]

M0 = f([5, 10])

Y = 0
N0 = bpt_node(x=4.5, y=Y, M=M0, widths=widths, height=height)

p += N0
print(p)
```



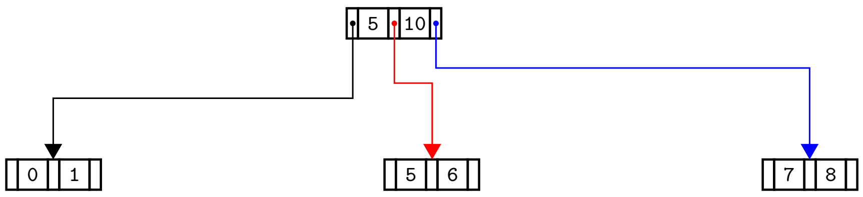
49.2 Directed edge

```
from latextool_basic import *
p = Plot()
widths= [0.15, 0.4]; height = 0.4

def f(M):
    return [r'{{\scriptsize\texttt{%s}}}' % _ for _ in M]

M0, M1, M2, M3 = f([5, 10]), f([0, 1]), f([5, 6]), f([7, 8])

N0 = bpt_node(x=4.5, y=0, M=M0, widths=widths, height=height)
N1 = bpt_node(x=0, y=-2, M=M1, widths=widths, height=height)
N2 = bpt_node(x=5, y=-2, M=M2, widths=widths, height=height)
N3 = bpt_node(x=10, y=-2, M=M3, widths=widths, height=height)

arc0 = bpt_arc(N0, N1, 0);
arc1 = bpt_arc(N0, N2, 1, linecolor='red')
arc2 = bpt_arc(N0, N3, 2, linecolor='blue')

p += N0; p += N1; p += N2; p += N3
p += arc0; p += arc1; p += arc2

print(p)
```



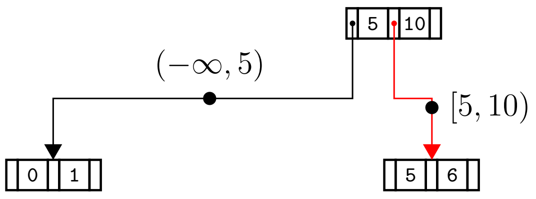
Changing height of the bend.

```
from latextool_basic import *
p = Plot()
widths= [0.15, 0.4]; height = 0.4

def f(M):
    return [r'{{\scriptsize\texttt{%s}}}' % _ for _ in M]

M0, M1, M2, M3 = f([5, 10]), f([0, 1]), f([5, 6]), f([7, 8])

N0 = bpt_node(x=4.5, y=0, M=M0, widths=widths, height=height)
N1 = bpt_node(x=0, y=-2, M=M1, widths=widths, height=height)
N2 = bpt_node(x=5, y=-2, M=M2, widths=widths, height=height)
N3 = bpt_node(x=10, y=-2, M=M3, widths=widths, height=height)

arc0 = bpt_arc(N0, N1, 0);
arc1 = bpt_arc(N0, N2, 1, linecolor='red', delta=0.2)
arc2 = bpt_arc(N0, N3, 2, linecolor='blue', delta=0.4)

p += N0; p += N1; p += N2; p += N3
p += arc0; p += arc1; p += arc2

print(p)
```



Labels

```
from latextool_basic import *
p = Plot()
widths= [0.15, 0.4]; height = 0.4

def f(M):
    return [r'{{\scriptsize\texttt{%s}}}' % _ for _ in M]

M0 = f([5, 10]); M1 = f([0, 1]); M2 = f([5, 6])

Y = 0
N0 = bpt_node(x=4.5, y=Y, M=M0, widths=widths, height=height)

Y = -2
N1 = bpt_node(x=0, y=Y, M=M1, widths=widths, height=height)
N2 = bpt_node(x=5, y=Y, M=M2, widths=widths, height=height)

arc0 = bpt_arc(N0, N1, 0); arc1 = bpt_arc(N0, N2, 1, linecolor='red')

p += N0; p += N1; p += N2
p += arc0; p += arc1

from latexcircuit import *
x,y = arc0.midpoint()
p += Circle(x=x, y=y, r=0.05, background='red')
X = POINT(x=x, y=y, label='$(-\infty,5)$', anchor='south')
p += str(X)

x,y = arc1.midpoint(ratio=0.7)
p += Circle(x=x, y=y, r=0.05, background='blue')
X = POINT(x=x, y=y, label='$[5, 10)$', anchor='west')
p += str(X)

print(p)
```

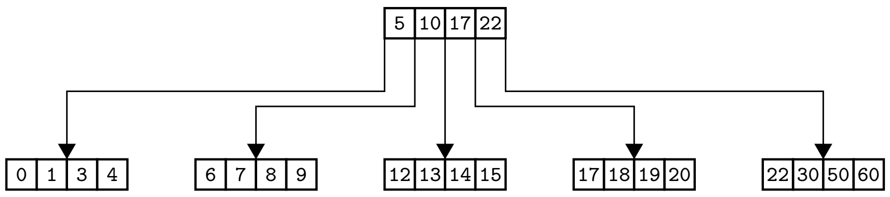


```
from latextool_basic import *
from latexcircuit import *

def f(M): return tuple(r'{{\scriptsize\texttt{%s}}}' % _ for _ in M)
```

```
p = Plot()
widths = [0.1, 0.4]
height = 0.4
node_sep = 0.1
vsep = 1.5 # vertical separation -- from bottom of root to top of child


keyss = [(10, 30, 65, 80), (1, 5, 7, 9), (10, 14, 15, 28), (35, 42, 45, 57),
         (68, 72, 75, 79), (80, 89, 90, 92)]
Ms = [f(keys) for keys in keyss]

Y = -(height + vsep)
children = bptree_get_siblings(0, Y, Ms[1:], widths, height, node_sep)
root = bptree_get_root(0, keyss[0], children, widths, height)

for _ in [root] + children: p += _

arcs = bptree_get_arcs(root, children, vsep, height)
for _ in arcs: p += _

labels = bptree_get_labels(keyss, arcs)
for label in labels:
    p += label




"""
            A
      B   C   D   E
     H I     F G
              J K
"""
edges = {'A':['B','C','D','E'],
         'D':['F','G'],
         'B':['H','I'],
         'G':['J','K'],
        }
nodes = {'A':f([0,1,2,3]),
         'B':f([4,'','','']),
         'C':f([8,9,10,11]),
         'D':f([12,'','','']),
         'E':f([16,17,18,19]),
         'F':f([20,21,22,23]),
         'G':f([24,24,24,'']),
         'H':f([28,29,30,31]),
         'I':f([32,33,34,35]),
         'J':f([36,37,38,39]),
         'K':f([40,41,42,43]),
         }
bptree(p, edges=edges, nodes=nodes)

print(p)
```

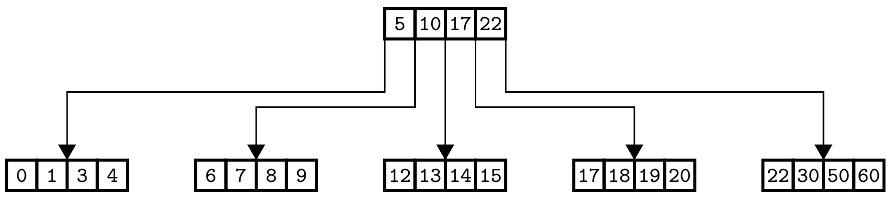
Width is 0 for pointers:

```
from latextool_basic import *
p = Plot()

widths= [0, 0.4]
height = 0.4

def f(M):
    return [r'{{\scriptsize\texttt{%s}}}' % _ for _ in M]

M0 = f([5, 10, 17, 22])
M1 = f([0, 1, 3, 4])
M2 = f([6, 7, 8, 9])
M3 = f([12, 13, 14, 15])
M4 = f([17, 18, 19, 20])
M5 = f([22, 30, 50, 60])

N0 = bpt_node(x=5, y=0, M=M0, widths=widths, height=height)

N1 = bpt_node(x=0, y=-2, M=M1, widths=widths, height=height)
N2 = bpt_node(x=2.5, y=-2, M=M2, widths=widths, height=height)
N3 = bpt_node(x=5, y=-2, M=M3, widths=widths, height=height)
N4 = bpt_node(x=7.5, y=-2, M=M4, widths=widths, height=height)
N5 = bpt_node(x=10, y=-2, M=M5, widths=widths, height=height)
p += N0; p += N1; p += N2; p += N3; p += N4; p+= N5

p += bpt_arc(N0, N1, 0, delta=0.2, r=0)
p += bpt_arc(N0, N2, 1, r=0)
p += bpt_arc(N0, N3, 2, r=0)
p += bpt_arc(N0, N4, 3, r=0)
p += bpt_arc(N0, N5, 4, delta=0.2, r=0)

print(p)
```



Width for keys is 0. Might be easier to read (when width of pointer is 0) if the linewidth of rects is thicker:

```
from latextool_basic import *
p = Plot()

widths= [0, 0.4]
height = 0.4

def f(M):
    return [r'{{\scriptsize\texttt{%s}}}' % _ for _ in M]

M0,M1,M2,M3,M4,M5 = (f([5, 10, 17, 22]), f([0, 1, 3, 4]), f([6, 7, 8, 9]),
          f([12, 13, 14, 15]), f([17, 18, 19, 20]), f([22, 30, 50, 60]))

def bpt_node_(x, y, M):
    return bpt_node(x=x, y=y, M=M, widths=widths, height=height,
                    linewidth=0.04)

N0 = bpt_node_(x=5, y=0, M=M0)
N1 = bpt_node_(x=0, y=-2, M=M1)
N2 = bpt_node_(x=2.5, y=-2, M=M2)
N3 = bpt_node_(x=5, y=-2, M=M3)
N4 = bpt_node_(x=7.5, y=-2, M=M4)
N5 = bpt_node_(x=10, y=-2, M=M5)
p += N0; p += N1; p += N2; p += N3; p += N4; p+= N5

p += bpt_arc(N0, N1, 0, delta=0.2, r=0)
p += bpt_arc(N0, N2, 1, r=0)
p += bpt_arc(N0, N3, 2, r=0)
p += bpt_arc(N0, N4, 3, r=0)
p += bpt_arc(N0, N5, 4, delta=0.2, r=0)

print(p)
```

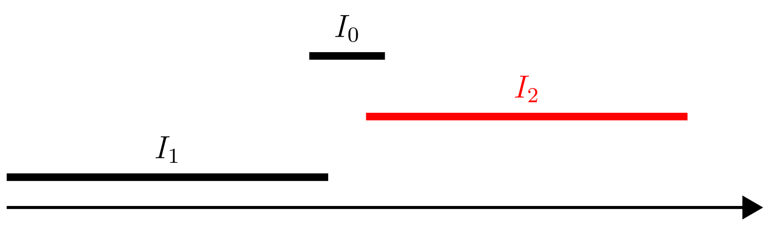


debug: interv

50 Intervals

For interval scheduling problems.

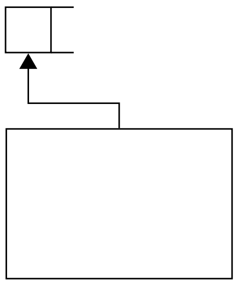
```
from latextool_basic import *
p = Plot()

intervals(p,
          dy=0.8,
          axislength=10,
          xss=[[(0, 4.25, '$I_1$'), (4.75, 9, '$I_2$', 'red')],
               [(4, 5, '$I_0$')]],
         )
print(p)
```



debug: dfa.te
chap-automat

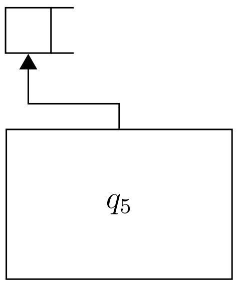
51 Automatas

```
from latextool_basic import *
p = Plot()

dfa(p)
print(p)
```



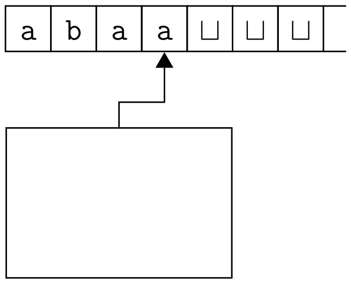
51.1 State

```
from latextool_basic import *
p = Plot()

dfa(p,
    state=r'$q_5$')
print(p)
```



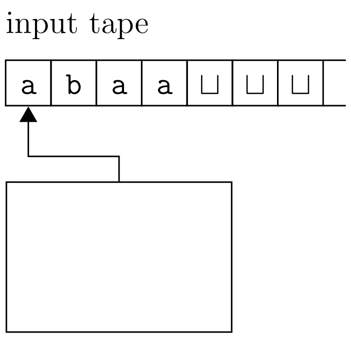
51.2 Tape and head

```
from latextool_basic import *
p = Plot()

dfa(p,
    tape=['a','b','a','a',r'\SPACE',r'\SPACE',r'\SPACE'],
    head_index=3,
    )
print(p)
```



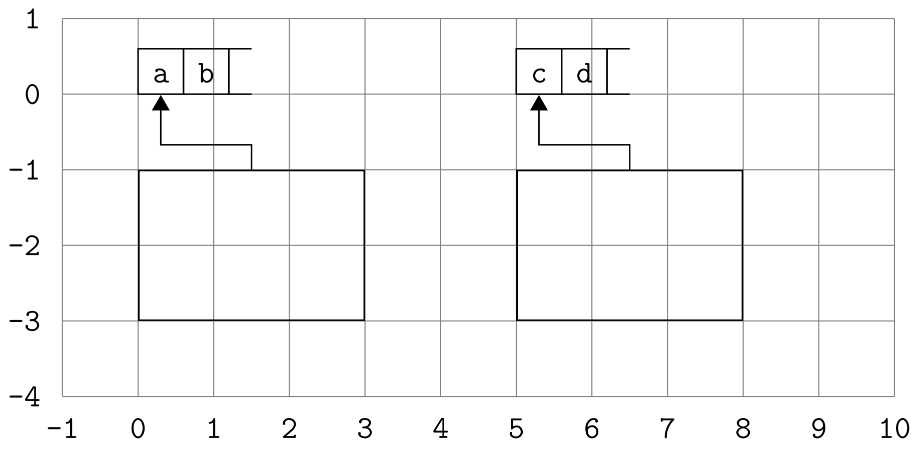
51.3 Label for tape

```
from latextool_basic import *
p = Plot()

dfa(p,
    tape=['a','b','a','a',r'\SPACE',r'\SPACE',r'\SPACE'],
    input_tape_str=True,
    )
print(p)
```



51.4 Position

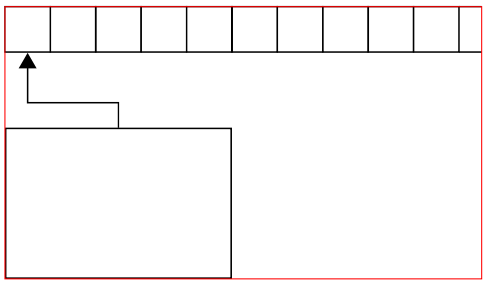
```
from latextool_basic import *
p = Plot()

dfa(p, x0=0, y0=0,
    tape=['a','b'],
    )
dfa(p, x0=5, y0=0,
    tape=['c','d'],
    )
p += Grid(x0=-1, y0=-4, x1=10, y1=1)
print(p)
```



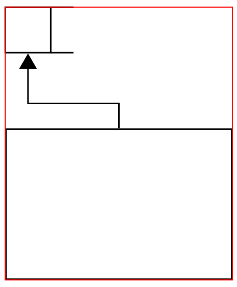
Bounding rect

The return value of `dfa` is a Rectangle object.

```
from latextool_basic import *
p = Plot()

r = dfa(p, tape=['' for i in range(10)],)
x0,y0 = r.bottomleft()
x1,y1 = r.topright()
p += Rect(x0=x0,y0=y0,x1=x1,y1=y1,linecolor='red')
print(p)
```

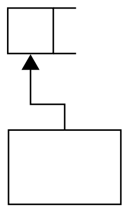


```
from latextool_basic import *
p = Plot()

r = dfa(p, tape=[''])
x0,y0 = r.bottomleft()
x1,y1 = r.topright()
p += Rect(x0=x0,y0=y0,x1=x1,y1=y1,linecolor='red')
print(p)
```



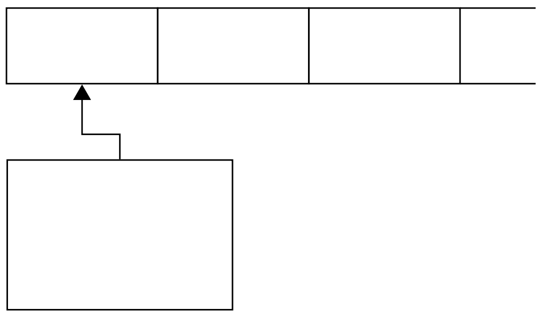
Dimensions of automata:

```
from latextool_basic import *
p = Plot()

r = dfa(p,
    body_w = 1.5,
    body_h = 1,
    )
print(p)
```



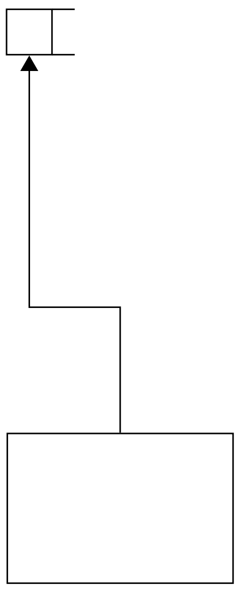
Cell width and height

```
from latextool_basic import *
p = Plot()

dfa(p,
    tape=['','',''],
    w=2, h=1,
    )
print(p)
```



51.5 Body-tape distance

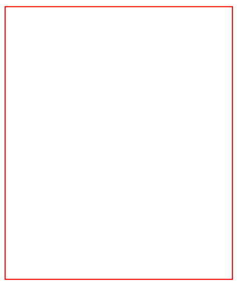
```
from latextool_basic import *
p = Plot()

dfa(p, vsep=5)
print(p)
```



51.6 No drawing

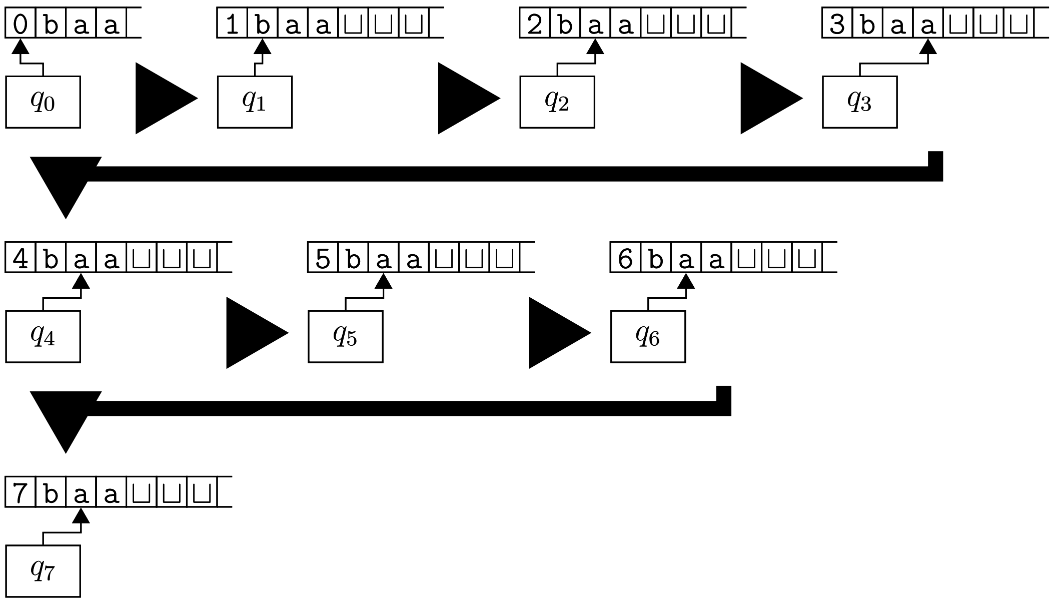
No drawing (just to compute bounding rect):

```
from latextool_basic import *
p = Plot()

r = dfa(p, no_draw=True)
x0,y0 = r.bottomleft()
x1,y1 = r.topright()
p += Rect(x0=x0, y0=y0, x1=x1, y1=y1, linecolor='red')
print(p)
```



51.7 Sequence of automatas

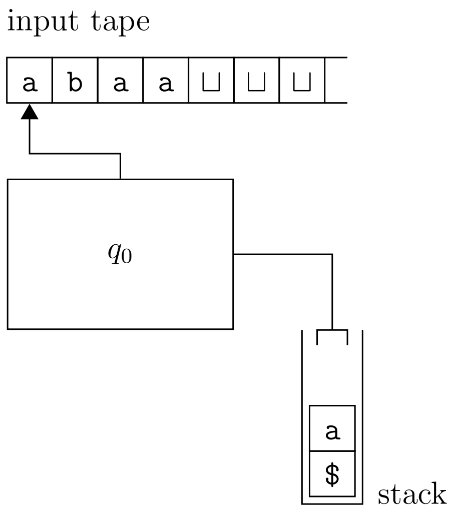
```
from latextool_basic import *
p = Plot()

def rect(index, p, x0, y0, no_draw=False):
    data = [(r'$q_0$',
              ['0','b','a','a'],
              0),
             (r'$q_1$',
              ['1','b','a','a',r'\SPACE',r'\SPACE',r'\SPACE'],
              1),
             (r'$q_2$',
              ['2','b','a','a',r'\SPACE',r'\SPACE',r'\SPACE'],
              2),
             (r'$q_3$',
              ['3','b','a','a',r'\SPACE',r'\SPACE',r'\SPACE'],
              3),
             (r'$q_4$',
              ['4','b','a','a',r'\SPACE',r'\SPACE',r'\SPACE'],
              2),
             (r'$q_5$',
              ['5','b','a','a',r'\SPACE',r'\SPACE',r'\SPACE'],
              2),
             (r'$q_6$',
              ['6','b','a','a',r'\SPACE',r'\SPACE',r'\SPACE'],
              2),
             (r'$q_7$',
              ['7','b','a','a',r'\SPACE',r'\SPACE',r'\SPACE'],
              2),
            ]
    if index < len(data):
        state, tape, head_index = data[index]
        return dfa(p,
               x0=x0,y0=y0,
               tape=tape,
               state=state,
               head_index=head_index,
               vsep=0.5, hsep=0.25,
               body_w=1, body_h=0.7, w=0.4,
               no_draw=no_draw,
               )
    else:
        return None

sequence(p, rect=rect)
print(p)
```



debug: pda.t

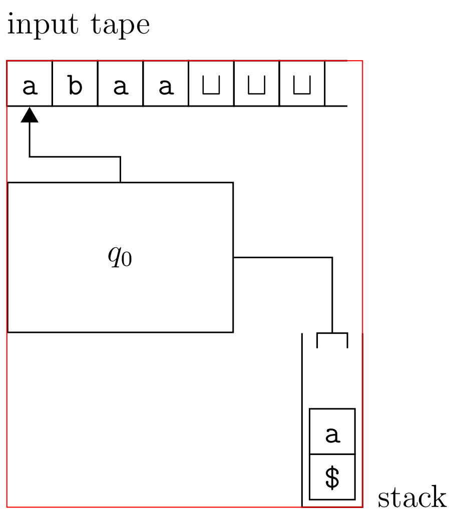
51.8 PDA

```
from latextool_basic import *
p = Plot()

pda(p,
    stackvalues = ['a','\$'],
    tape = ['a','b','a','a',r'\SPACE',r'\SPACE',r'\SPACE'],
    state = r'$q_0$',
    head_index = 0,
    body_w = 3,
    body_h = 2,
    vsep = 1,
    hsep = 1,
    input_tape_str=True,
    stack_str=True,
    )
print(p)
```


input tape
a
b
a
a
⊔
⊔
⊔
q_0
a
$
stack

51.9 PDA: bounding rect

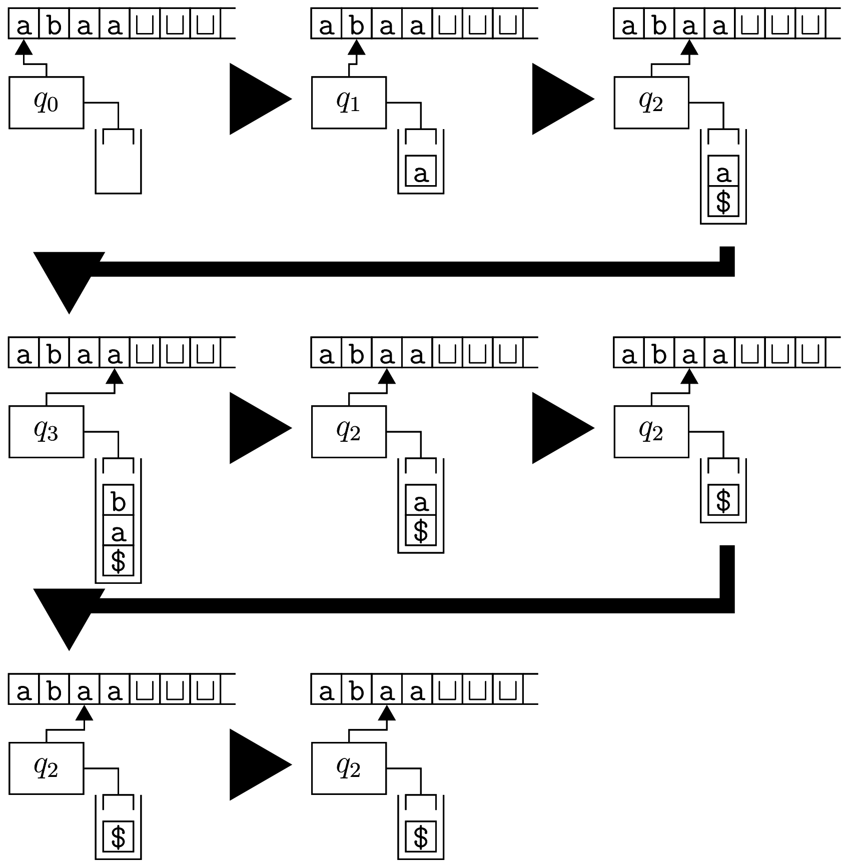
```
from latextool_basic import *
p = Plot()

r = pda(p,
    stackvalues = ['a','\$'],
    tape = ['a','b','a','a',r'\SPACE',r'\SPACE',r'\SPACE'],
    state = r'$q_0$',
    head_index = 0,
    body_w = 3,
    body_h = 2,
    vsep = 1,
    hsep = 1,
    input_tape_str=True,
    stack_str=True,
    )
x0,y0 = r.bottomleft()
x1,y1 = r.topright()
p += Rect(x0=x0,y0=y0,x1=x1,y1=y1,linecolor='red')
print(p)
```



```
from latextool_basic import *
p = Plot()

pda_computation(p,
    data = [(r'$q_0$',
             ['a','b','a','a',r'\SPACE',r'\SPACE',r'\SPACE'],
             0,
             []),
            (r'$q_1$',
             ['a','b','a','a',r'\SPACE',r'\SPACE',r'\SPACE'],
             1,
             ['a',]), # 'b','c', 'd', 'e', r'\$']),
            (r'$q_2$',
             ['a','b','a','a',r'\SPACE',r'\SPACE',r'\SPACE'],
             2,
             ['a', r'\$']),
            (r'$q_3$',
             ['a','b','a','a',r'\SPACE',r'\SPACE',r'\SPACE'],
             3,
             ['b', 'a', r'\$']),
             (r'$q_2$',
             ['a','b','a','a',r'\SPACE',r'\SPACE',r'\SPACE'],
             2,
             ['a', r'\$']),
            (r'$q_2$',
             ['a','b','a','a',r'\SPACE',r'\SPACE',r'\SPACE'],
             2,
             [r'\$']),
            (r'$q_2$',
             ['a','b','a','a',r'\SPACE',r'\SPACE',r'\SPACE'],
             2,
             [r'\$']),
            (r'$q_2$',
             ['a','b','a','a',r'\SPACE',r'\SPACE',r'\SPACE'],
             2,
             [r'\$']),
            ]
    )

print(p)
```

51.10 Sequence of PDA

```
from latextool_basic import *
p = Plot()

def rect(index, p, x0, y0, no_draw=False):
    data = [(r'$q_0$',
             ['a','b','a','a',r'\SPACE',r'\SPACE',r'\SPACE'],
             0,
             []),
            (r'$q_1$',
             ['a','b','a','a',r'\SPACE',r'\SPACE',r'\SPACE'],
             1,
             ['a',]), # 'b','c', 'd', 'e', r'\$']),
            (r'$q_2$',
             ['a','b','a','a',r'\SPACE',r'\SPACE',r'\SPACE'],
             2,
             ['a', r'\$']),
            (r'$q_3$',
             ['a','b','a','a',r'\SPACE',r'\SPACE',r'\SPACE'],
             3,
             ['b', 'a', r'\$']),
            (r'$q_2$',
             ['a','b','a','a',r'\SPACE',r'\SPACE',r'\SPACE'],
             2,
             ['a', r'\$']),
            (r'$q_2$',
             ['a','b','a','a',r'\SPACE',r'\SPACE',r'\SPACE'],
             2,
             [r'\$']),
            (r'$q_2$',
             ['a','b','a','a',r'\SPACE',r'\SPACE',r'\SPACE'],
             2,
             [r'\$']),
            (r'$q_2$',
             ['a','b','a','a',r'\SPACE',r'\SPACE',r'\SPACE'],
             2,
             [r'\$']),
            ]

    if index < len(data):
        state, tape, head_index, stackvalues = data[index]
        return pda(p,
                   x0=x0,y0=y0,
                   stackvalues = stackvalues,
                   tape= tape,
                   state = state,
                   head_index = head_index,
                   vsep=0.5, hsep=0.25,
                   body_w=1, body_h=0.7, w=0.4,
                   no_draw=no_draw,
                   )
    else:
        return None

sequence(p, rect=rect)
print(p)
```

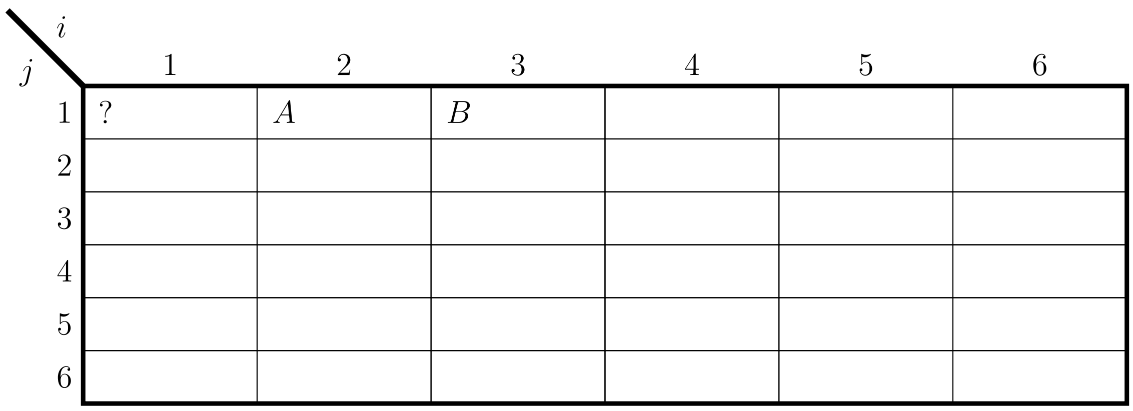


a b a a ⊔ ⊔ ⊔
q_0
q_1
q_2
q_3
a
$
b

debug: cyk.te

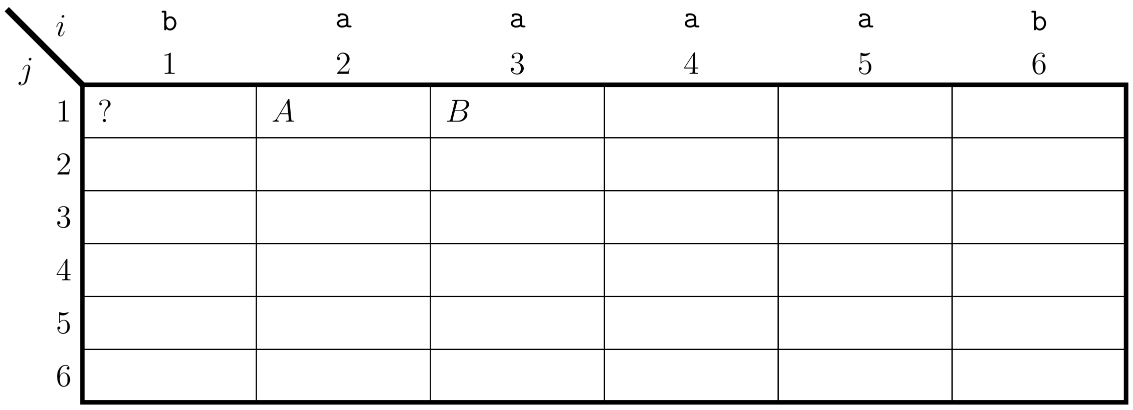
51.11 CYK

```
from latextool_basic import *
m = [['?','$A$','$B$','','',''],
     ['','','','','',''],
     ['','','','','',''],
     ['','','','','',''],
     ['','','','','',''],
     ['','','','','',''],
     ]
p = Plot()
cyk(p, m)
print(p)
```



51.11.1 Word being parsed

```
from latextool_basic import *
m = [['?','$A$','$B$','','',''],
     ['','','','','',''],
     ['','','','','',''],
     ['','','','','',''],
     ['','','','','',''],
     ['','','','','',''],
     ]
p = Plot()
cyk(p, m, w='baaaab')
print(p)
```

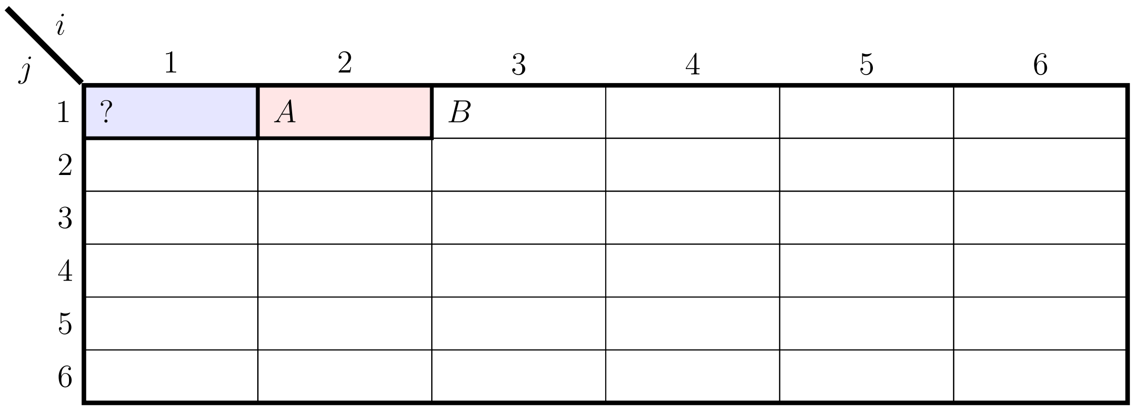


j \ i	`b` 1	`a` 2	`a` 3	`a` 4	`a` 5	`b` 6
1	?	A	B			
2						
3						
4						
5						
6						

51.11.2 Backgound

You can set the background using the (row, col) index values. If it's set, the cell is also drawn with a larger linewidth.

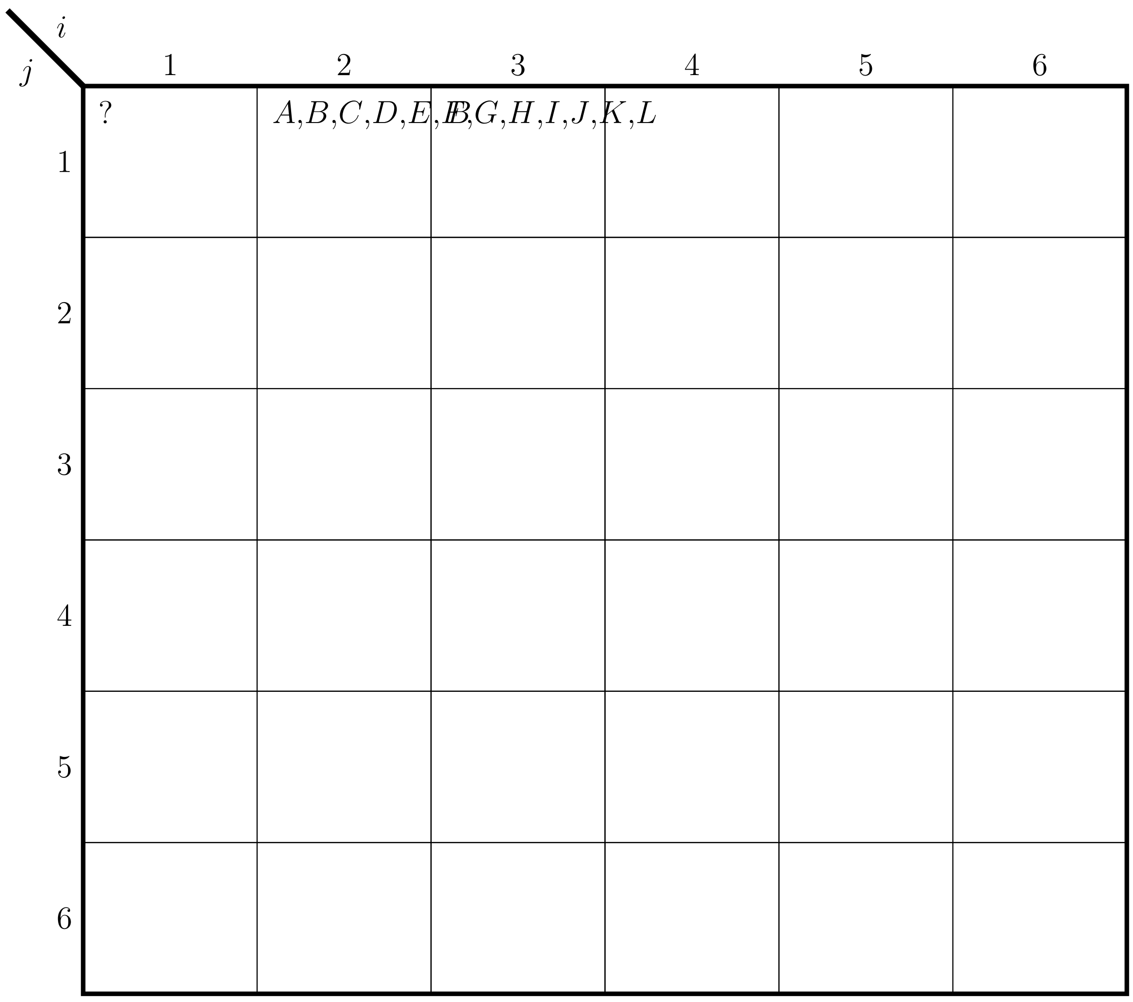
```
from latextool_basic import *
m = [['?','$A$','$B$','','',''],
     ['','','','','',''],
     ['','','','','',''],
     ['','','','','',''],
     ['','','','','',''],
     ['','','','','',''],
     ]
p = Plot()
cyk(p, m, background={(0,0):'blue!10',
                      (0,1):'red!10',})
print(p)
```



j \ i	1	2	3	4	5	6
1	?	A	B			
2						
3						
4						
5						
6						

51.11.3 Height and width

```
from latextool_basic import *
m = [['?','$A$,$B$,$C$,$D$,$E$,$F$,$G$,$H$,$I$,$J$,$K$,$L$','$B$','','','']
,
     ['','','','','',''],
     ['','','','','',''],
     ['','','','','',''],
     ['','','','','',''],
     ['','','','','',''],
     ]
p = Plot()
cyk(p, m, height=2)
print(p)
```



debug: auton

52 Automata

The `automata` function (in the `latextool_basic` module) spits out latex code for state diagrams which can then be modified by hand.

Here are some examples.

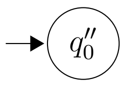
If you execute the following python code:

```
print(automata(layout="""
    A
    """,
    A="initial|label=$q''_0$",
    ))
```

you will get this latex code as output:

```
\begin{center}
\begin{tikzpicture}[shorten >=1pt,node distance=2cm,auto,initial text=]
\node[state,initial] (A) at (  4,  0) {$q''_0$};

\path[->]


;
\end{tikzpicture}
\end{center}
```

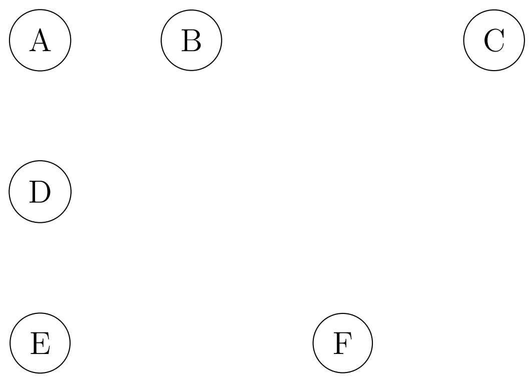
When you paste the above output into a latex document, you will get this picture:



The `layout` parameter is a string for placing the nodes. When you execute this:

```
from latextool_basic import *
automata(layout="""
A B   C
D
E   F
"""
)
```

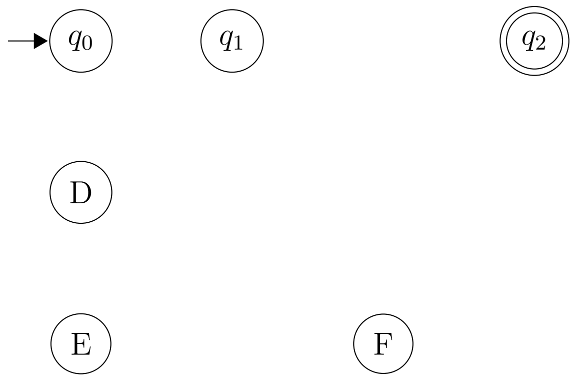
you get this output:



You can then put decorate each node.

```
from latextool_basic import *
print(automata(layout="""
A B   C
D
E   F
""",
A = "initial|label=$q_0$",
B = "label=$q_1$",
C = "accept|label=$q_2$"
))
```

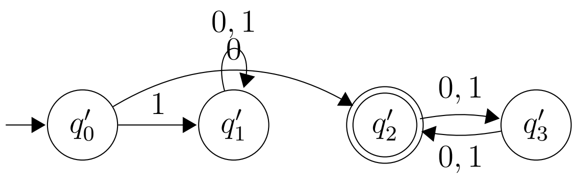
gives this picture



For the rest of the examples, I will just give you the python code and the corresponding picture.

DFA for “starts with 0 and has odd length”:

```
from latextool_basic import *
print(automata(layout="""
A D B C
""",
edges="A,$0$,B|B,$0,1$,C|C,$0,1$,B|A,$1$,D|D,$0,1$,D",
A="initial|label=$q'_0$",
B="accept|label=$q'_2$",
C="label=$q'_3$",
D="label=$q'_1$",
))
```

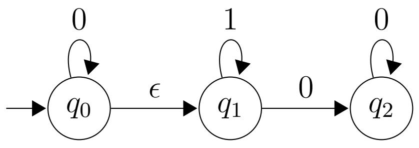


Sipser 1.7c

Sipser 1.7d

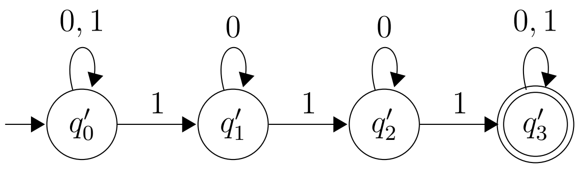
Sipser 1.7e

```
from latextool_basic import *
print(automata(layout="""
A B C
""",
edges="A,$\ep$,B|A,$0$,A|B,$1$,B|B,$0$,C|C,$0$,C",
A="initial|label=$q_0$",
B="label=$q_1$",
C="label=$q_2$",
))
```



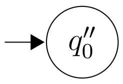
NFA for at least three 1s:

```
from latextool_basic import *
print(automata(layout="""
A B C D
""",
edges="A,1,B|B,1,C|C,1,D|A,$0,1$,A|D,$0,1$,D|B,0,B|C,0,C",
A="initial|label=$q'_0$",
B="label=$q'_1$",
C="label=$q'_2$",
D="accept|label=$q'_3$",
))
```



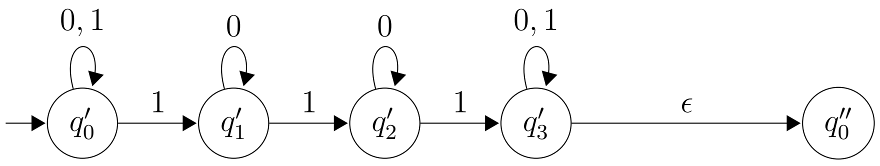
NFA for $\emptyset$:

```
from latextool_basic import *
print(automata(layout="""
A
""",
A="initial|label=$q''_0$",
))
```



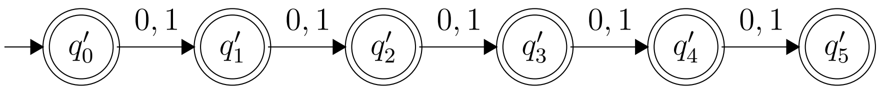
NFA for concat of "contains at least 3 1s" and $\emptyset$:

```
from latextool_basic import *
print(automata(layout="""
A B C D   E
""",
edges="A,1,B|B,1,C|C,1,D|A,$0,1$,A|D,$0,1$,D|B,0,B|C,0,C|D,$\ep$,E",
A="initial|label=$q'_0$",
B="label=$q'_1$",
C="label=$q'_2$",
D="label=$q'_3$",
E="label=$q''_0$"
))
```



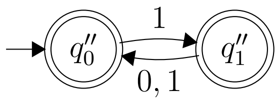
NFA for concat of “length <= 5” and “every odd position is a 1”:

```
from latextool_basic import *
print(automata(layout="""
A B C D E F
""",
edges="A,$0,1$,B|B,$0,1$,C|C,$0,1$,D|D,$0,1$,E|E,$0,1$,F",
A="accept|initial|label=$q'_0$",
B="accept|label=$q'_1$",
C="accept|label=$q'_2$",
D="accept|label=$q'_3$",
E="accept|label=$q'_4$",
F="accept|label=$q'_5$",
))
```



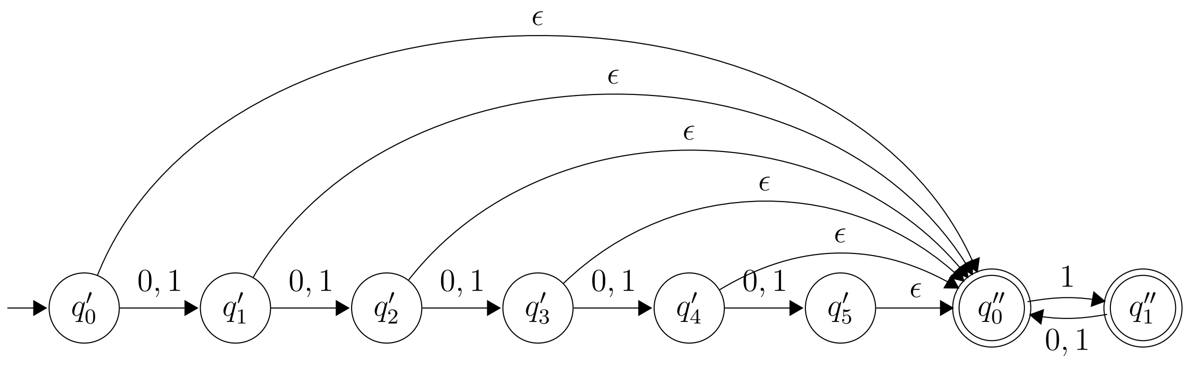
NFA for “every odd position is a 1”:

```
from latextool_basic import *
print(automata(layout="""
A B
""",
edges="A,$1$,B|B,$0,1$,A",
A="accept|initial|label=$q''_0$",
B="accept|label=$q''_1$",
))
```



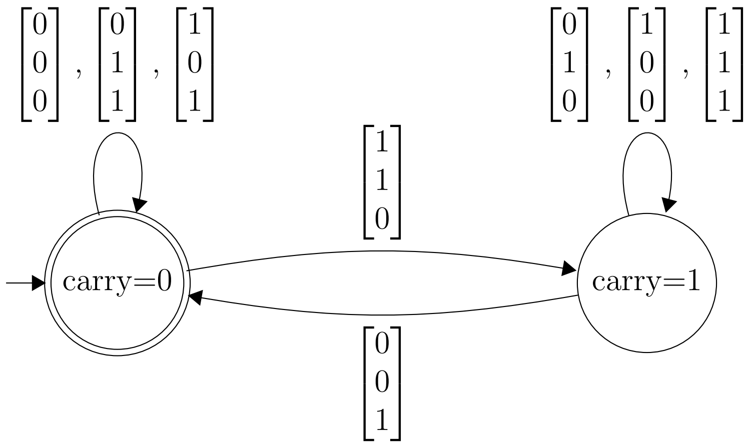
NFA for concat of “length <=5” and “every odd position is a 1”:

```
from latextool_basic import *
print(automata(layout="""
A B C D E F G H
""",
edges="A,$0,1$,B|B,$0,1$,C|C,$0,1$,D|D,$0,1$,E|E,$0,1$,F"+\
 "|F,$\epsilon$,G"+\
 "|A,$\epsilon$,G"+\
 "|B,$\epsilon$,G"+\
 "|C,$\epsilon$,G"+\
 "|D,$\epsilon$,G"+\
 "|E,$\epsilon$,G"+\
 "|G,$1$,H|H,$0,1$,G",
A="initial|label=$q'_0$",
B="label=$q'_1$",
C="label=$q'_2$",
D="label=$q'_3$",
E="label=$q'_4$",
F="label=$q'_5$",
G="accept|label=$q''_0$",
H="accept|label=$q''_1$",
))
```



NFA with vectors as Σ:

```
from latextool_basic import *
print(automata(layout="""
A      B
""",
edges=r"A,$\begin{bmatrix}0\\0\\0 \end{bmatrix},\begin{bmatrix}0\\1\\1 \end
{bmatrix},\begin{bmatrix}1\\0\\1 \end{bmatrix}$,A"+\
r"|A,$\begin{bmatrix}1\\1\\0\end{bmatrix}$,B"+\
r"|B,$\begin{bmatrix}0\\1\\0\end{bmatrix},\begin{bmatrix}1\\0\\0\end{bmatri
x},\begin{bmatrix}1\\1\\1\end{bmatrix}$,B"+\
r"|B,$\begin{bmatrix}0\\0\\1\end{bmatrix}$,A"+\
r"",
A="initial|accept|label=carry=0",
B="label=carry=1",
))
```


$\begin{bmatrix}0\\0\\0\end{bmatrix},\begin{bmatrix}0\\1\\1\end{bmatrix},\begin{bmatrix}1\\0\\1\end{bmatrix}$
$\begin{bmatrix}0\\1\\0\end{bmatrix},\begin{bmatrix}1\\0\\0\end{bmatrix},\begin{bmatrix}1\\1\\1\end{bmatrix}$
$\begin{bmatrix}1\\1\\0\end{bmatrix}$
carry=0
carry=1
$\begin{bmatrix}0\\0\\1\end{bmatrix}$

debug: longta
minipage.tex

53 Longtable of minipages

abc	def
`v.size()`	return size of `v` etc. etc. etc. etc. etc. etc. etc. etc. etc. etc. etc. etc. etc. etc. etc. etc. etc. etc. etc. etc. etc. etc. etc. etc.
`v.size()`	return size of `v` abc def
abc def def def def def def def def def	return size of `v` abc def

debug: soluti

54 Solutions

This is not part of latextool.

See `projects/solutions`

Put symlink `/usr/lib64/python3.7/site-packages/solutions.py` to point to `projects/solutions/`

How to use: In a section (or chapter) of latex notes with exercises put

```
\section{Natural numbers $\N$}
\begin{python0}
from solutions import *; clear()
\end{python0}

\begin{ex}
  \label{ex:succ-onto-nonzero}
  What is 1 + 1?
  \solutionlink{sol:succ-onto-nonzero}
  \qed
\end{ex}
\begin{python0}
label = "succ-onto-nonzero"
s = r'''
\proof
1 + 1 = 2
\qed
'''
from solutions import *; add(label, s)
\end{python0}

\begin{prop}
  \label{prop:N-is-commutative-monoid}
  What is 1 + 1?
\end{prop}

\proof
Exercise.
\solutionlink{sol:N-is-commutative-monoid}
\qed
\begin{python0}
from solutions import *
add(label="prop:N-is-commutative-monoid",
    srcfilename='proof-N-is-commutative-monoid.tex')
\end{python0}

\newpage
\subsection{Solutions}

\input{solutions.tex}
```