Wolfram *Mathematica*® Tutorial Collection

GRIDS, ROWS, AND COLUMNS IN *MATHEMATICA*

For use with Wolfram *Mathematica*® 7.0 and later.

For the latest updates and corrections to this manual:
visit reference.wolfram.com

For information on additional copies of this documentation:
visit the Customer Service website at www.wolfram.com/services/customerservice
or email Customer Service at info@wolfram.com

Comments on this manual are welcomed at:
comments@wolfram.com

Printed in the United States of America.

15 14 13 12 11 10 9 8 7 6 5 4 3 2

Contents

Grids, Rows, and Columns in *Mathematica*

The Basic Constructs

Mathematica provides a broad range of powerful constructs for laying out content on a screen or page. They are designed to be immediately useful for the beginner, yet also allow fine control over almost every aspect of their appearance.

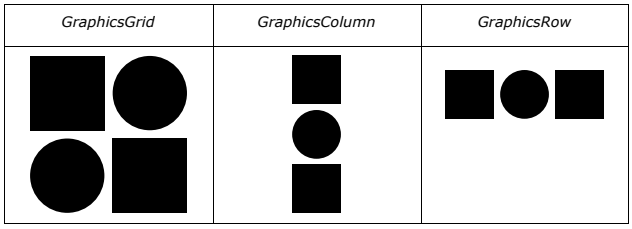
These constructs can be placed into three families: constructs that appear within notebooks as typesetting structures, functions that generate graphics whose contents are arranged on a grid, and constructs that can appear inside grids to adjust details of formatting.

`Grid`, `Column`, and `Row` form the first family, referred to in this tutorial as the Grid family. The Grid family's defining characteristic is that it is a tightly integrated part of *Mathematica*'s typesetting system. This means that any expression whatsoever can appear as content, and that the construct itself can respond to changes such as window width or even the size of its elements. Like other typesetting constructs, the Grid family's members are inert constructs and do not evaluate to some other form.

Grid	*Column*	*Row*
a b c d e f g h i j k l m n o p q r s t u v w x	a b c d	abcdefghijklm nopqrstuvwxyz

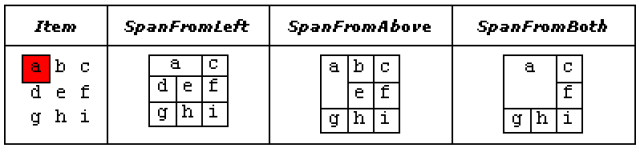
Examples of the Grid family of formatting constructs.

A parallel set of constructs—the Graphics Grid family—supports features particularly useful when dealing with graphics. These constructs are `GraphicsGrid`, `GraphicsColumn`, and `GraphicsRow`. Though graphics can be used in the Grid family, the Graphics Grid family supports sizing and editing behavior more tailored to graphics. The Graphics Grid family has functions that take arguments and evaluate them into new graphics expressions, which means it is difficult to make the generated grid respond to changes in its environment, but easy to interactively add arbitrary annotation and additional graphics.



Examples of the Graphics Grid family of formatting constructs.

The final family—the Embedded Constructs family—consists of constructs that are embedded within the grids themselves, and alter the grid's appearance from within. `Item` can be wrapped around elements in a grid in order to style the region in which they appear. `SpanFromLeft`, `SpanFromAbove`, and `SpanFromBoth` are used to create regions that span across multiple rows or columns.



Examples of embeddable constructs.

Some very basic examples of all of these constructs follow.

Grid Family

`Grid[{{`$expr_{11}$`,`$expr_{12}$`,…},{`$expr_{21}$`,`$expr_{22}$`,…},…}]`	an object that formats with the $expr_{ij}$ arranged in a two-dimensional grid
`Column[{`$expr_1$`,`$expr_2$`,…}]`	an object that formats with the $expr_i$ arranged in a column, with $expr_1$ above $expr_2$, etc.
`Row[{`$expr_1$`,`$expr_2$`,…}]`	an object that formats with the $expr_i$ arranged in a row, potentially extending over several lines

The Grid family of 2D formatting constructs.

A grid of elements.

```
In[684]:= Grid[{{"a", "b", "c"}, {"d", "e", "f"}, {"g", "h", "i"}}]

Out[684]= a b c
          d e f
          g h i
```

A column of elements.

```
In[686]:= Column[{"a", "b", "c", "d"}]
Out[686]= a
          b
          c
          d
```

A row of elements.

```
In[685]:= Row[CharacterRange["a", "z"]]
Out[685]= abcdefghijklmnopqrstuvwxyz
```

Graphics Grid Family

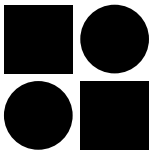
GraphicsGrid[{{g_{11},g_{12},…},…}]	generates a graphic in which the g_{ij} are laid out in a two-dimensional grid
GraphicsColumn[{g_1,g_2,…}]	generates a graphic in which the g_i are laid out in a column, with g_1 above g_2, etc.
GraphicsRow[{g_1,g_2,…}]	generates a graphic in which the g_i are laid out in a row

The Graphics Grid family of 2D graphics layout functions.

Display elements in a graphics grid.

```
In[678]:= GraphicsGrid[{{■, ●}, {●, ■}}]
```

Out[678]=



Display elements in a graphics column.

```
In[689]:= GraphicsColumn[{■, ●, ■}]
```

Out[689]=

Display elements in a graphics row.

```
In[691]:= GraphicsRow[{■, ●, ■}]
```

Out[691]=

Embedded Constructs Family

`Item[`*expr*`,`*options*`]`	displays with *expr* as the content, and with the specified options applied to the region containing *expr*
`SpanFromLeft`	indicates that the position is occupied by the contents on the left
`SpanFromAbove`	indicates that the position is occupied by the contents above
`SpanFromBoth`	indicates that the position is occupied from both above and left

Constructs with special meaning when they are embedded as elements within the Grid and Graphics Grid families.

Embed styling information around the element `"a"`.

```
In[708]:= Grid[{{Item["a", Background -> Red, Frame -> True], "b", "c"},
   {"d", "e", "f"}, {"g", "h", "i"}}]
```

Out[708]=

a	b	c
d	e	f
g	h	i

Span `"a"` across the first two columns.

```
In[709]:= Grid[{{"a", SpanFromLeft, "c"}, {"d", "e", "f"}, {"g", "h", "i"}}, Frame -> All]
```

Out[709]=

a		c
d	e	f
g	h	i

Span `"a"` across the first two rows.

```
In[710]:= Grid[{{"a", "b", "c"}, {SpanFromAbove, "e", "f"}, {"g", "h", "i"}},
  Frame -> All]
```

Out[710]=

a	b	c
	e	f
g	h	i

Span `"a"` across the first two columns and rows.

```
In[711]:= Grid[{{"a", SpanFromLeft, "c"},
            {SpanFromAbove, SpanFromBoth, "f"}, {"g", "h", "i"}}, Frame -> All]
```

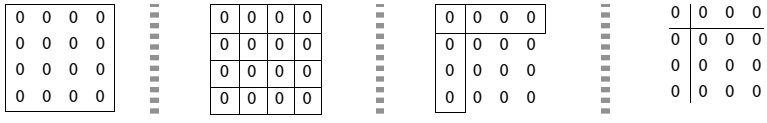
Out[711]=

a		c
		f
g	h	i

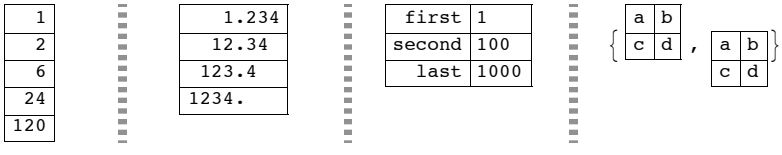
Classes of Functionality

`Grid` and related constructs allow considerable appearance customization, often with very little syntax. The tables below indicate the kind of functionality that is supported; it is explained in detail in later sections.

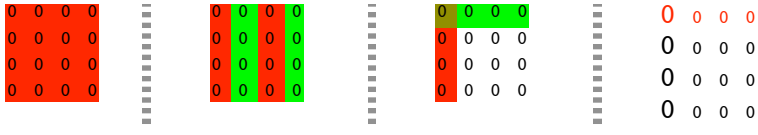
- Frames and dividers can be placed in any position to partition regions of the grid.



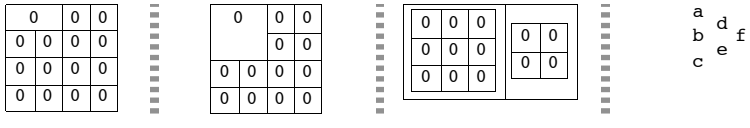
- The grid and its contents can be aligned and positioned in a number of ways.



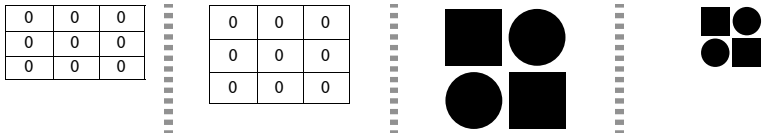
- Backgrounds and styles can be imposed on any region.



- The structure can be elaborated using spanning elements, or elements that are themselves grids.



- The sizes and spacings in the grid can be adjusted.



In addition to these styling features, various forms of interactive editing and dynamic behavior are possible.

Options Syntax

A variety of options exist for adjusting the details of a grid's appearance. This section describes the common syntax shared by many of these options. This syntax provides a way to assign option values not only for the entire grid, but also for individual rows, columns, and even items.

The overall syntax for many options, such as `Background`, is based on forms like `Background -> {`$spec_x$`, `$spec_y$`}`, where $spec_x$ is itself a modular syntax that contains values for different columns, while $spec_y$ contains values for the different rows.

$spec$	apply $spec$ to all items
$\{spec_x\}$	apply $spec_x$ at successive horizontal positions
$\{spec_x, spec_y\}$	apply $spec_k$ at successive horizontal and vertical positions
$\{spec_x, spec_y, rules\}$	give rules for the items based on their i, j position in the array

General options syntax.

$spec_x$ and $spec_y$ may take two possible forms, as described below. The first form is just the rules for the desired value at a set of indices. The second form is based on giving a sequence of values in a list.

A set of rules specifying the index of a column and its desired background.

```
In[8]:= Grid[{{1, 2, 3}, {4, 5, 6}, {7, 8, 9}},
         Background -> {{1 → Red, 2 → Green, 3 → Blue}, Automatic}]
```

Out[8]=

An equivalent list of background values to use for successive columns.

```
In[9]:= Grid[{{1, 2, 3}, {4, 5, 6}, {7, 8, 9}},
         Background -> {{Red, Green, Blue}, Automatic}]
```

Out[9]=

1	2	3
4	5	6
7	8	9

These two methods have different strengths, as described in "Using Rules" and "Using Lists".

Using Rules

Rules provide a direct and readable method to give a specific row or column a specific value.

Set the background for specific sets of columns.

```
In[16]:= Grid[Table[x, {4}, {4}], Background -> {{1 → Red, 3 → Red}, Automatic}]
```

Out[16]=

x	x	x	x
x	x	x	x
x	x	x	x
x	x	x	x

Set the background for specific sets of rows.

```
In[26]:= Grid[Table[x, {4}, {4}], Background -> {Automatic, {1 → Red, 3 → Red}}]
```

Out[26]=

x	x	x	x
x	x	x	x
x	x	x	x
x	x	x	x

When there is a large number of rows or columns, rules are a convenient way to set the properties of just a few of them.

Apply the option at a small number of the possible positions.

```
In[25]:= Grid[Table[x, {4}, {10}], Background -> {{1 → Red, 3 → Red, -2 → Red}, Automatic}]
```

Out[25]=

x	x	x	x	x	x	x	x	x	x
x	x	x	x	x	x	x	x	x	x
x	x	x	x	x	x	x	x	x	x
x	x	x	x	x	x	x	x	x	x

Rules can also be used to give values to specific grid elements or subregions. Note however that while conceptually similar, this following syntax is separate from the discussion of $spec_x$ and $spec_y$.

Set the background of the element at position {3, 3}.

```
In[27]:= Grid[Table[x, {4}, {10}], Background -> {Automatic, Automatic, {3, 3} → Red}]
```

Out[27]=

Set the background of the region from element {1, 1} to {3, 3}.

```
In[28]:= Grid[Table[x, {4}, {10}],
          Background -> {Automatic, Automatic, {{1, 3}, {1, 3}} → Red}]
```

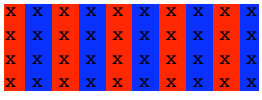
Out[28]=

Rules are an efficient way to specify the exceptions to the value that otherwise exists. However, they are less efficient when the intention is to manually specify a value for each piece of the grid.

Manually specify an alternating pattern using rules.

```
In[31]:= Grid[Table[x, {4}, {10}],
          Background -> {{1 → Red, 2 → Blue, 3 → Red, 4 → Blue, 5 → Red,
              6 → Blue, 7 → Red, 8 → Blue, 9 → Red, 10 → Blue}, Automatic}]
```

Out[31]=



To achieve repetitive patterns, it is instead recommended to use the list syntax described in the next section.

Using Lists

Giving sequential values in a list is a compact and convenient way to specify large numbers of option values for adjacent rows or columns.

List the values to be used for successive columns.

```
In[52]:= Grid[Table[x, {4}, {10}],
          Background -> {{Red, Green, Blue, Orange, Yellow}, Automatic}]
```

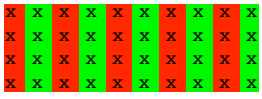
Out[52]=

Additionally, sublists can be used to denote cyclic use of values.

Specify that the list should be used cyclically.

```
In[53]:= Grid[Table[x, {4}, {10}], Background -> {{{Red, Green}}, Automatic}]
```

Out[53]=



These cyclic sublists can be padded at the beginning or end.

Give an initial set of values before the cyclic portion.

```
In[58]:= Grid[Table[x, {4}, {10}], Background -> {{Blue, Orange, {Red, Green}}, Automatic}]
```

Out[58]=

Give a final set of values.

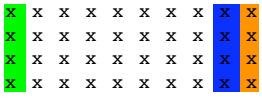
```
In[57]:= Grid[Table[x, {4}, {10}], Background -> {{{Red, Green}, Blue, Orange}, Automatic}]
```

Out[57]=

Use defaults in the middle.

```
In[7]:= Grid[Table[x, {4}, {10}],
          Background -> {{Green, {Automatic}, Blue, Orange}, Automatic}]
```

Out[7]=

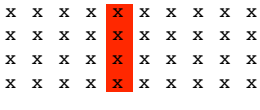


Because positions in the list correspond to positions in the grid, specifying a single value in the middle requires giving all previous values. To achieve this more directly, use rules as described in the section "Using Rules".

Make the fifth column red using the list syntax.

```
In[61]:= Grid[Table[x, {4}, {10}],
          Background -> {{Automatic, Automatic, Automatic, Automatic, Red}, Automatic}]
```

Out[61]=



Use a rule to directly assign the background.

```
In[62]:= Grid[Table[x, {4}, {10}], Background -> {{5 → Red}, Automatic}]
```

```
Out[62]= x x x x x x x x x x
         x x x x x x x x x x
         x x x x x x x x x x
         x x x x x x x x x x
```

Using Both

It is possible to have the best of both worlds, using the list syntax for specifying repetitive portions of the grid while also using the rule syntax to specify exceptions.

Columns alternate between blue and green, except the first and last, which are red.

```
In[746]:= Grid[Table[x, {4}, {10}],
           Background -> {{{{Blue, Green}}, {1 → Red, -1 → Red}}, Automatic}]
```

```
Out[746]= x x x x x x x x x x
          x x x x x x x x x x
          x x x x x x x x x x
          x x x x x x x x x x
```

Use blue for all columns, except the first and fifth.

```
In[89]:= Grid[Table[x, {4}, {10}], Background -> {{Blue, {1 → Red, 5 → Red}}, Automatic}]
```

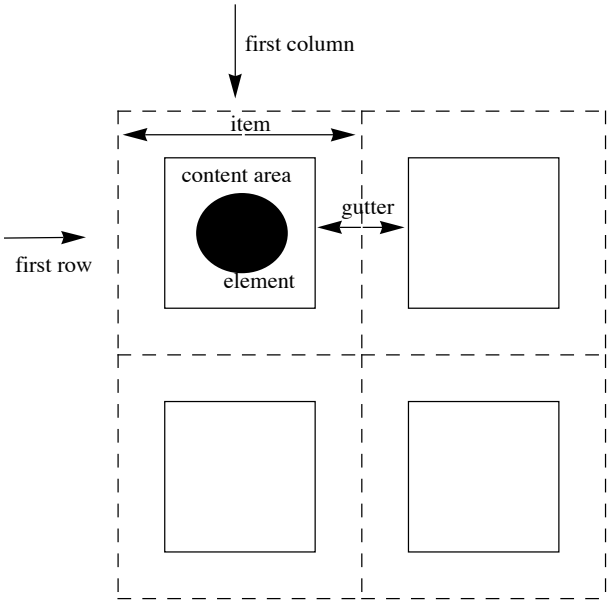
```
Out[89]= x x x x x x x x x x
         x x x x x x x x x x
         x x x x x x x x x x
         x x x x x x x x x x
```

$\{s_1, s_2, \ldots, s_n\}$	use s_1 through s_n; then use defaults
$\{\{c\}\}$	use c in all cases
$\{\{c_1, c_2\}\}$	alternate between c_1 and c_2
$\{\{c_1, c_2, \ldots\}\}$	cycle through all c_i
$\{s, \{c\}\}$	use s, then repeatedly use c
$\{s_1, \{c\}, s_n\}$	use s_1, then repeatedly use c, but use s_n at the end
$\{s_1, s_2, \ldots, \{c_1, c_2, \ldots\}, s_m, \ldots, s_n\}$	use the first sequence of s_i at the beginning, then cyclically use the c_i, then use the last sequence of s_i at the end
$\{s_1, s_2, \ldots, \{\}, s_m, \ldots, s_n\}$	use the first sequence of s_i at the beginning and the last sequence at the end
$\{i_1 \to v_1, i_2 \to v_2, \ldots\}$	specify what to use at positions i_k
$\{spec, rules\}$	use *rules* to override specifications in *spec*

Summary of syntax for $spec_x$ and $spec_y$.

Columns, Rows, Gutters, and Items

As introduced in the previous section, *Mathematica* provides a flexible syntax for changing an option's value in different regions of a grid. This section provides context for that language and elaborates on the finer distinctions.



Vocabulary for grids.

`column`	vertical sequence of items
`row`	horizontal sequence of items
`item`	the region containing a grid element
`gutter`	the border between consecutive rows or columns

Different slices of a 2D grid.

`Grid` and `GraphicsGrid` follow the same conventions for describing the different possible slices of the grid. `Column`, `GraphicsColumn`, and `GraphicsRow` follow the same general conventions, except that they only deal in one of the two possible dimensions. Finally, `Row` does not participate in this system at all.

The World of Options

The following table identifies the slices each listed option can address. No option is valid for all constructs; refer to the key below to see which option can occur for a given construct.

Out[54]=

Option	entire grid	rows columns	items	gutters
Alignment	☑	☑	☑	☐
AlignmentPoint	☑	☐	☐	☐
AspectRatio	☑	☐	☐	☐
Background	☑	☑	☑	☐
BaselinePosition	☑	☐	☑	☐
BaseStyle	☑	☐	☐	☐
DefaultBaseStyle	☑	☐	☐	☐
DefaultElement	☑	☐	☐	☐
Dividers	☑	☐	☐	☑
Frame	☑	☑	☑	☐
FrameStyle	☑	☐	☐	☐
ImageSize	☑	☐	☐	☐
ItemAspectRatio	☑	☐	☐	☐
ItemSize	☑	☑	☐	☐
ItemStyle	☑	☑	☑	☐
Spacings	☑	☐	☐	☑

Out[53]=

Key to option colors

Grid, Column, GraphicsGrid, GraphicsColumn, GraphicsRow
GraphicsGrid, GraphicsColumn, GraphicsRow
Grid, Column

Note that `Row` does not take any options.

Columns, Then Rows

To remember the syntax for options, the most important step is knowing that specific values for the columns are specified first, and values for rows are specified second.

opt->*val*	use *val* for all items
opt->*colspec*,*rowspec*}	use *colspec* for columns, *rowspec* for rows
opt->{*colspec*}	use *colspec* for columns, defaults for rows

Option structure for `Grid` and `GraphicsGrid`.

In *Mathematica,* options with a horizontal setting h and a vertical setting v are specified as *opt* -> {h, v}. `ImageSize` and `PlotRange` are two common options that help establish this convention.

In a grid, these horizontal and vertical settings correspond to values for the columns and rows, respectively. This is because columns are stacked horizontally, and so their properties—such as width and location—correspond to the horizontal dimension. Rows are stacked vertically, and their properties correspond to the vertical dimension.

A graphic that is twice as long as it is tall.

```
In[37]:= Graphics[{}, ImageSize → {100, 50}, Background → Black]
```

Out[37]=

In the following grid each item is 2 "ems" wide and 1 "ex" tall.

```
In[1]:= Grid[{{1, 2, 3}, {4, 5, 6}, {7, 8, 9}}, ItemSize → {2, 1}, Frame → All]
```

Out[1]=

1	2	3
4	5	6
7	8	9

Instead of a single width for all columns, a separate setting is given for each column.

```
In[29]:= Grid[{{1, 2, 3}, {4, 5, 6}, {7, 8, 9}}, ItemSize -> {{1, 2, 3}, 1}, Frame → All]
```

Out[29]=

1	2	3
4	5	6

```
In[2]:= Grid[{{1, 2, 3}, {4, 5, 6}, {7, 8, 9}},
  ItemSize -> {1, {1, 2, 3}}, Frame -> All]
```

Out[2]=

1	2	3
4	5	6
7	8	9

Similarly, a different background color can be given at successive horizontal positions.

```
In[6]:= Grid[{{1, 2, 3}, {4, 5, 6}, {7, 8, 9}},
         Background -> {{Red, Green, Blue}, Automatic}]
```

Out[6]=

1	2	3
4	5	6
7	8	9

Gutters

Many `Grid` options deal with properties that can ultimately be associated with a column, row, or item in the grid.

However there are also options that deal with gutters between rows and columns.

`Dividers`	where to draw divider lines in the grid
`Spacings`	horizontal and vertical spacings

Options for the gutters between rows and columns.

A line that is not associated with any single row or column.

```
In[85]:= Grid[Table[x, {3}, {13}], Dividers → {None, {2 → True}}]
```

Out[85]=

x x x x x x x x x x x x x
x x x x x x x x x x x x x
x x x x x x x x x x x x x

Compare this with a frame.

```
In[86]:= Grid[Table[x, {3}, {13}], Frame -> {None, {2 -> True}}]
```

Out[86]=

x x x x x x x x x x x x x
x x x x x x x x x x x x x
x x x x x x x x x x x x x

The syntax for `Dividers` and `Spacings` is exactly the same as for the other options. For a grid with n items in a particular direction, `Dividers` and `Spacings` can specify settings for the $n+1$ gaps between elements, starting before the first element and ending after the last element.

Items

The most granular level of description is the item. Each item in a grid can have its own value for options such as `Background`, `Alignment`, and `Frame`.

`Item` can be used to explicitly indicate the desired settings.

```
In[100]:= Grid[{{Item[a, Background → Red], b}, {c, d}}]
```

```
Out[100]= a b
          c d
```

Alternatively, use the item's $\{i, j\}$ index to assign it a value at the overall grid level.

```
In[101]:= Grid[{{a, b}, {c, d}}, Background → {Automatic, Automatic, {{1, 1} → Red}}]
```

```
Out[101]= a b
          c d
```

A programmatically generated grid.

```
In[103]:= Grid[Table[x, {5}, {5}],
           Background → {Automatic, Automatic, Table[{i, i} → Red, {i, 1, 5}]}]
```

```
Out[103]= x x x x x
          x x x x x
          x x x x x
          x x x x x
          x x x x x
```

Give settings to an entire region of the grid.

```
In[105]:= Grid[Table[x, {5}, {5}],
           Background → {Automatic, Automatic, {{{2, 4}, {2, 4}} → Red}}]
```

```
Out[105]= x x x x x
          x x x x x
          x x x x x
          x x x x x
          x x x x x
```

```
In[109]:= Grid[Table[x, {5}, {5}], Frame → {None, None, {{{2, 4}, {2, 4}} → True}}]
```

```
Out[109]= x x x x x
          x x x x x
          x x x x x
          x x x x x
          x x x x x
```

Dividers and Frames

Mathematica provides an extensive system for describing what dividers and frames should be drawn in a grid.

`Dividers`	draw dividers between columns or rows
`Frame`	put a frame around regions of the grid
`FrameStyle`	use an overall style for the lines

Options for drawing dividers and frames.

Use `Frame` to put lines on all four sides of a region or set of regions.

```
In[131]:= Grid[Table[x, {4}, {7}], Frame → True]
```

```
Out[131]=
x x x x x x x
x x x x x x x
x x x x x x x
x x x x x x x
```

Highlight specific columns or rows.

```
In[141]:= Grid[Table[x, {4}, {7}], Frame → {2 → True, 3 → True}]
```

```
Out[141]=
x x x x x x x
x x x x x x x
x x x x x x x
x x x x x x x
```

`Frame` always draws a line on all four faces of the enclosed region. `Dividers` allows a finer level of control.

```
In[140]:= Grid[Table[x, {4}, {7}], Dividers → {2 → True}]
```

```
Out[140]=
x | x x x x x x
x | x x x x x x
x | x x x x x x
x | x x x x x x
```

One result is that when using dividers, the resulting lines run in only a single direction.

```
In[147]:= Grid[Table[x, {4}, {7}], Dividers → {All, None}]
```

```
Out[147]=
x | x | x | x | x | x | x
x | x | x | x | x | x | x
x | x | x | x | x | x | x
x | x | x | x | x | x | x
```

```
In[146]:= Grid[Table[x, {4}, {7}], Frame → {All, None}]
```

```
Out[146]=
x | x | x | x | x | x | x
x | x | x | x | x | x | x
x | x | x | x | x | x | x
x | x | x | x | x | x | x
```

This short form draws the center dividers.

```
In[150]:= Grid[Table[x, {4}, {7}], Dividers -> Center]
```

Out[150]=

x	x	x	x	x	x	x
x	x	x	x	x	x	x
x	x	x	x	x	x	x
x	x	x	x	x	x	x

Styling Dividers and Frames

`FrameStyle` sets the base style used for both `Frame` and `Dividers`.

```
In[94]:= Grid[Table[x, {4}, {7}], Dividers → {All, None}, FrameStyle → Red]
```

Out[94]=

x	x	x	x	x	x	x
x	x	x	x	x	x	x
x	x	x	x	x	x	x
x	x	x	x	x	x	x

```
In[95]:= Grid[Table[x, {4}, {7}], Frame → {All, None}, FrameStyle → Red]
```

Out[95]=

x	x	x	x	x	x	x
x	x	x	x	x	x	x
x	x	x	x	x	x	x
x	x	x	x	x	x	x

`Frame` and `Dividers` allow styles as values.

```
In[50]:= Grid[Table[x, {4}, {7}], Dividers → {{Brown, {1 → Red, -1 → Green}}, None}]
```

Out[50]=

x	x	x	x	x	x	x
x	x	x	x	x	x	x
x	x	x	x	x	x	x
x	x	x	x	x	x	x

```
In[48]:= Grid[Table[x, {4}, {7}], Frame → {{Brown, {1 → Red, -1 → Green}}, None}]
```

Out[48]=

x	x	x	x	x	x	x
x	x	x	x	x	x	x
x	x	x	x	x	x	x
x	x	x	x	x	x	x

In general, any line and color directive may be used, including `Hue`, `Thickness`, `Dashing`, `Dotted`, and others. Multiple directives may be combined with `Directive`.

Precedence

When conflicting styles are given, `Dividers` has precedence over `Frame`, and they have precedence over `FrameStyle`. Styles from `Item` take precedence over all others.

Dividers and frames are added together.

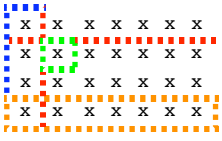
```
In[177]:= Grid[Table[x, {4}, {7}], Dividers → {2 → True, 3 → True}, Frame → {None, 4 → True}]
```

```
Out[177]=
| x | x x x x x x |
| x | x x x x x x |
|---|-------------|
| x | x x x x x x |
|---|-------------|
| x | x x x x x x |
```

Styles from different sources are combined together.

```
In[185]:= Grid[{{x, x, x, x, x, x, x}, {x, Item[x, Frame → Green], x, x, x, x, x},
            {x, x, x, x, x, x, x}, {x, x, x, x, x, x, x}},
           Dividers → {2 → Red, 2 → Red}, Frame → {{1 → True}, 4 → Orange},
           FrameStyle → Directive[Dashing[2], Thickness[2], Blue]]
```

Out[185]=



Alignment and Positioning

Aesthetic grids often require use of alignment. *Mathematica* has considerable support for different kinds of alignment in grids.

The `Alignment` option can be passed to the overall grid.

Align contents to the right.

```
In[53]:= Grid[{{1, 10}, {100, 1000}, {10 000, 100 000}}, Alignment → Right]
```

```
Out[53]=
     1       10
   100     1000
10 000  100 000
```

It is possible to give different horizontal alignments to different columns, and different vertical alignments to different rows.

Align the first column to the right, and the second column to the left.

```
In[61]:= Grid[{{1, 10}, {100, 1000}, {10 000, 100 000}}, Alignment → {{Right, Left}}]
```

```
Out[61]=
     1 10
   100 1000
10 000 100 000
```

It is also possible to give different alignments to the individual items in the grid.

Set the element at position `{1, 1}` to the left, and the element at position `{1, 2}` to the right.

```
In[62]:= Grid[{{1, 10}, {100, 1000}, {10 000, 100 000}},
          Alignment -> {{Right, Left}, Automatic, {{1, 1} → Left, {1, 2} → Right}}]
```

```
Out[62]=  1          10
            100  1000
          10 000  100 000
```

`Alignment` can also be set with `Item`. The specification for `Item` will take precedence.

```
In[63]:= Grid[{{Item[1, Alignment → Left], Item[10, Alignment → Right]},
           {100, 1000}, {10 000, 100 000}}, Alignment → {{Right, Left}}]
```

```
Out[63]=  1          10
            100  1000
          10 000  100 000
```

It is possible to align on a decimal point, or any character.

Align on ".".

```
In[65]:= Column[{1234., 123.4, 12.34, 1.234}, Alignment → "."]
```

```
Out[65]= 1234.
          123.4
           12.34
            1.234
```

Positioning a grid within its enclosing environment can be achieved with `BaselinePosition`.

Default position.

```
In[67]:= {a, Grid[{{1, 2}, {3, 4}}], b}
```

```
Out[67]= {a, 1  2
             3  4 , b}
```

Align the bottom of the grid to the baseline of the enclosing expression.

```
In[73]:= {a, Grid[{{1, 2}, {3, 4}}, BaselinePosition → Bottom, Frame → True], b}
```

```
Out[73]= {a, ┌─┬─┐
             │1│2│
             ├─┼─┤
             │3│4│
             └─┴─┘ , b}
```

Align the grid so that the baseline of the `{2, 1}` element is at the overall baseline.

```
In[74]:= {a, Grid[{{1, 2}, {3, 4}}, BaselinePosition → {2, 1}, Frame → True], b}
```

```
Out[74]= {a, ┌─┬─┐
             │1│2│
             ├─┼─┤
             │3│4│
             └─┴─┘ , b}
```

Background and Style

Common Cases

When working with a collection of elements, `Grid` provides a way to set them against a uniform background.

Though an element can have its own background, awkward gaps result when you put elements together.

```
In[193]:= Grid[{{Style[a, Background → Brown], Style[b, Background → Brown]}}]
Out[193]= a b
```

`Grid` and related functions place a background across the entire group of items in which the elements are contained.

```
In[194]:= Grid[{{a, b}}, Background → Brown]
Out[194]= a b
```

With more sophisticated syntax, a variety of patterns is easy to achieve.

Alternate the backgrounds.

```
In[195]:= Grid[Table[x, {5}, {7}], Background -> {{{Brown, None}}}]
Out[195]= x x x x x x x
          x x x x x x x
          x x x x x x x
          x x x x x x x
          x x x x x x x

In[196]:= Grid[Table[x, {5}, {7}], Background -> {None, {{Brown, None}}}]
Out[196]= x x x x x x x
          x x x x x x x
          x x x x x x x
          x x x x x x x
          x x x x x x x
```

Highlight a row and column that intersect.

```
In[197]:= Grid[Table[x, {5}, {7}], Background -> {5 → Brown, 4 → Brown}]
Out[197]= x x x x x x x
          x x x x x x x
          x x x x x x x
          x x x x x x x
          x x x x x x x
```

Embed the background with a particular item inside the grid.

```
In[107]:= Grid[{{"The ", "quick ", Item["brown", Background → Brown],
              " fox ", "jumps ", "over ", "the ", "lazy ", "dog."}}]
Out[107]= The quick brown fox jumps over the lazy dog.
```

Precedence of Overlapping Background Settings

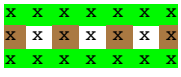
Backgrounds given in the list syntax blend together upon intersection.

```
In[198]:= Grid[Table[x, {3}, {7}], Background → {{{Brown, None}}, {{Green, None}}}]
```

```
Out[198]=
```

Backgrounds specifically asserted using indices take precedence.

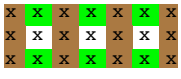
```
In[199]:= Grid[Table[x, {3}, {7}], Background → {{{Brown, None}}, {1 → Green, 3 → Green}}]
```



```
Out[199]=
```

Indexed columns take precedence over indexed rows.

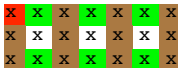
```
In[200]:= Grid[Table[x, {3}, {7}],
           Background → {{1 → Brown, 3 → Brown, 5 → Brown, 7 → Brown}, {{Green, None}}}]
```



```
Out[200]=
```

Backgrounds specified with `Item` have the highest precedence.

```
In[201]:= Grid[{{Item[x, Background → Red], x, x, x, x, x, x},
             {x, x, x, x, x, x, x}, {x, x, x, x, x, x, x}},
           Background → {{1 → Brown, 3 → Brown, 5 → Brown, 7 → Brown}, {{Green, None}}}]
```



```
Out[201]=
```

Spanning and Nesting

Sophisticated partitioning of 2D space can be achieved by nested grid constructs and/or by using spanning elements.

As their name suggests, spanning elements allow an item to span multiple columns, rows, or both.

Span "a" across the first two columns.

```
In[709]:= Grid[{{"a", SpanFromLeft, "c"}, {"d", "e", "f"}, {"g", "h", "i"}}, Frame -> All]
```

Out[709]=

a		c
d	e	f
g	h	i

Span "a" across the first two rows.

```
In[710]:= Grid[{{"a", "b", "c"}, {SpanFromAbove, "e", "f"}, {"g", "h", "i"}},
           Frame -> All]
```

Out[710]=

a	b	c
	e	f
g	h	i

Span "a" across the first two columns and rows.

```
In[711]:= Grid[{{"a", SpanFromLeft, "c"},
            {SpanFromAbove, SpanFromBoth, "f"}, {"g", "h", "i"}}, Frame -> All]
```

Out[711]=

a		c
		f
g	h	i

It is important to note that the spanning region must be rectangular; items that fail to fall within the rectangle will not be spanned, and will instead display the spanning character.

Spanning is only done in rectangular chunks.

```
In[75]:= Grid[{{"a", SpanFromLeft, "c"},
           {SpanFromAbove, SpanFromBoth, SpanFromLeft}, {"g", "h", "i"}}, Frame -> All]
```

Out[75]=

a		c
		…
g	h	i

While many layouts can be achieved using spanning elements, it is sometimes faster or more convenient to simply nest grid constructs.

```
In[81]:= Row[{Column[{a, b, c}], Column[{d, e}], f}]
```

```
Out[81]= a d
         b   f
         c e
```

```
In[84]:= Grid[{{Grid[{{a, b}, {c, d}}], e}, {f, g}}]
```

```
Out[84]= a b
         c d   e
          f    g
```

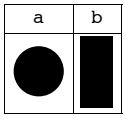
Particularly with complex grids, it is often clearer to use `Row` and `Column` to create the specifically desired structures than to try to design a complicated system of spanning.

Sizing and Spacing

Sizing in Grid

`Grid` will typically not modify the size of its elements. Also, rows and columns are by default made as narrow as possible while accommodating the contents.

```
In[275]:= Grid[{{a, b}, {●, ▮}}, Frame → All]
```



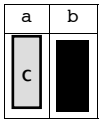
Out[275]=

Notice in the above example that the second row is much taller than the first, the second column thinner than the first, and the sizes of the elements were not modified in any way.

If elements in the grid are interactively or dynamically changed, the size of the entire grid will automatically adjust as appropriate.

A useful exception is that `Button` will by default expand to fill the available space.

```
In[277]:= Grid[{{a, b}, {Button[c], ▮}}, Frame → All]
```



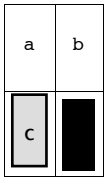
Out[277]=

`ItemSize` can be used to override the default behavior.

Make all items the same size.

```
In[278]:= Grid[{{a, b}, {Button[c], }}, Frame → All, ItemSize → All]
```

Out[278]=



Specify widths and heights for individual columns and rows.

```
In[256]:= Grid[Table[x, {3}, {7}], Frame → All, ItemSize → {{1, 2, 3, 4, 5, 6, 7}, 1}]
```

Out[256]=

x	x	x	x	x	x	x
x	x	x	x	x	x	x
x	x	x	x	x	x	x

The units used for `ItemSize` are the typesetting units known as "exs" and "ems."

It is also possible to specify widths as a fraction of the enclosing area by using `Scaled`.

Make the first two columns each .3 of the page width.

```
In[274]:= Grid[Table[x, {3}, {7}], Frame → All,
  ItemSize → {{Scaled[.3], Scaled[.3], 3, 4, 5, 6, 7}, 1}]
```

Out[274]=

x	x	x	x	x	x	x
x	x	x	x	x	x	x
x	x	x	x	x	x	x

Line Wrapping in Grid

Textual items will line wrap if the columns are too narrow. Notice that this forces the rows to be taller than the minimum specified.

```
In[266]:= Grid[Table[xxxx, {3}, {7}], Frame → All, ItemSize → {{1, 2, 3, 4, 5, 6, 7}, 1}]
```

Out[266]=

x x x x	xx x x	xxxx	xxxx	xxxx	xxxx	xxxx
x x x x	xx x x	xxxx	xxxx	xxxx	xxxx	xxxx
x x x x	xx x x	xxxx	xxxx	xxxx	xxxx	xxxx

With `ItemSize -> Automatic`, textual items are wrapped at the page width.

```
In[271]:= Grid[{{100 !}, {200 !}}, Frame → All, ItemSize → Automatic]
```

Out[271]=

93 326 215 443 944 152 681 699 238 856 266 700 490 715 968 264 381 621 468 592 963 895 217 599 993 229 915 608 941 463 976 156 518 286 253 697 920 827 223 758 251 185 210 916 864 000 000 000 000 000 000 000 000
788 657 867 364 790 503 552 363 213 932 185 062 295 135 977 687 173 263 294 742 533 244 359 449 963 403 342 920 304 284 011 984 623 904 177 212 138 919 638 830 257 642 790 242 637 105 061 926 624 952 829 931 113 462 857 270 763 317 237 396 988 943 922 445 621 451 664 240 254 033 291 864 131 227 428 294 853 277 524 242 407 573 903 240 321 257 405 579 568 660 226 031 904 170 324 062 351 700 858 796 178 922 222 789 623 703 897 374 720 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000

With `ItemSize -> Full`, line breaking is prevented.

```
In[273]:= Grid[{{100 !}, {110 !}}, Frame → All, ItemSize → Full]
```

Out[273]=

93 326 215 443 944 152 681 699 238 856 266 700 490 715 968 264 381 621 468 592 963 895 217 599 993 229
15 882 455 415 227 429 404 253 703 127 090 772 871 724 410 234 473 563 207 581 748 318 444 567 162 948 183 030 959 9

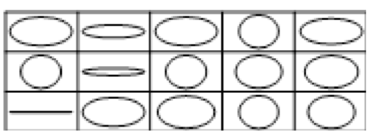
Sizing in GraphicsGrid

`GraphicsGrid` will by default return a grid whose items are all the same size.

```
In[286]:= randomgrid = Table[Graphics[Circle[], ImageSize → Automatic,
           AspectRatio -> RandomReal[]], {3}, {5}];

In[289]:= GraphicsGrid[randomgrid, Frame → All]
```

Out[289]=

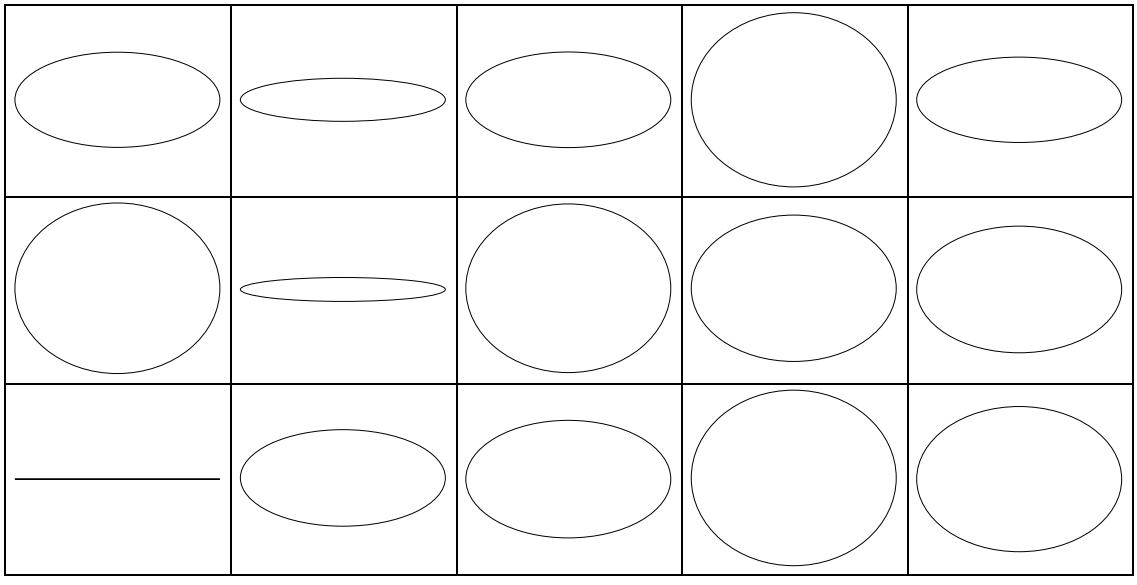


It will automatically choose an aspect ratio that is appropriate for the overall collection of elements.

Compare this with the equivalent `Grid` example, which does not impose either an overall size or an aspect ratio.

```
In[290]:= Grid[randomgrid, Frame → All]
```

Out[290]=



`GraphicsGrid` does not support an `ItemSize` option, but it does support `ImageSize`.

```
In[294]:= GraphicsGrid[randomgrid, ImageSize → 200, Frame → All]
```

Out[294]=