Overview

When using the `grid` geometry manager, widgets are placed into cells formed by rows and columns. In many layouts, a widget ends up being **smaller than the cell it is placed in**, which leads to an important question: where inside the cell should the widget appear?

This lesson introduces the `sticky` option, which controls how widgets are positioned and stretched within their grid cells. It also explains how `sticky` works together with **row and column weights** to allow windows and widgets to resize correctly.

Important Information

Default Widget Placement in a Cell

By default, when a widget is smaller than the grid cell it occupies, grid centers the widget both horizontally and vertically within that cell. Any extra space around the widget shows the background of the parent container.

This default behavior is fine for simple layouts, but it often causes alignment problems in forms and structured interfaces where widgets need to line up along edges.

The Sticky Option

The `sticky` option changes how a widget is positioned **inside its grid cell**. Instead of always centering the widget, `sticky` allows you to attach the widget to one or more sides of the cell.

Sticky uses compass directions to describe positioning:

- `n` for top (north)
- `s` for bottom (south)
- `e` for right (east)
- `w` for left (west)

```
label.grid(row=0, column=0, sticky="n")
```

In this example, the widget is pushed to the top of the cell while remaining centered horizontally.

Combining Sticky Directions

Sticky directions can be combined to achieve more precise positioning. For example, using `"nw"` sticks the widget to the top-left corner of the cell. Any extra space remains on the bottom and right sides.

```
button.grid(row=1, column=0, sticky="nw")
```

Using combinations like this is common when aligning labels, buttons, or icons in a structured layout.

Stretching Widgets with Sticky

Sticky can also be used to stretch widgets. When opposite directions are specified, grid stretches the widget to fill the space between those edges.

Using `"we"` stretches a widget horizontally, while `"ns"` stretches it vertically.

```
entry.grid(row=0, column=1, sticky="we")
```

This is especially useful for entry widgets, which often need to expand to match the width of their column.

Filling the Entire Cell

To make a widget expand in all directions and completely fill its cell, use all four compass directions.

```
frame.grid(row=0, column=0, sticky="nsew")
```

This causes the widget to stick to every side of the cell and occupy all available space. This pattern is extremely common when placing frames inside a window, since frames are often used as containers for other widgets.

Sticky vs Widget Content Alignment

It is important to distinguish between how a widget is positioned inside a cell and how content is positioned inside the widget itself.

The `sticky` option controls **where the widget sits inside the grid cell**. Widget-specific options, such as `anchor` on a label, control **where the widget's content is placed inside the widget**.

```
label = ttk.Label(root, text="Hello", anchor="w")
label.grid(row=0, column=0, sticky="nsew")
```

Here, `sticky="nsew"` stretches the label to fill the entire cell, while `anchor="w"` aligns the text to the left within the label.

Resizing Widgets with the Window

You may notice that even when using `sticky="nsew"`, widgets do not resize when the window itself is resized. This happens because grid rows and columns do not expand by default.

Each row and column has a `weight` value that controls whether it can grow when extra space is available. The default weight is `0`, which means no expansion occurs. Sticky only tells grid how to behave *if* a cell grows; it does not cause the cell to grow.

Row and Column Weights

Row and column weights determine how extra space is distributed. A weight of `0` prevents expansion, while any positive value allows expansion. Larger values cause a row or column to grow faster relative to others.

Weights are assigned using the `columnconfigure` and `rowconfigure` methods.

The following example assumes all widgets are placed inside a frame called `content`. The weights are applied to that frame's grid.

```
content.columnconfigure(0, weight=3)
content.columnconfigure(1, weight=3)
content.columnconfigure(2, weight=3)
content.columnconfigure(3, weight=1)
content.columnconfigure(4, weight=1)

content.rowconfigure(1, weight=1)
```

In this layout, columns 0 through 2 grow three times faster than columns 3 and 4. Row 1 is allowed to grow vertically. These values are relative, not absolute.

For this to work correctly, the root window must also allow the frame to expand.

```
root.columnconfigure(0, weight=1)
root.rowconfigure(0, weight=1)
```

Without this step, the frame itself will not receive extra space, even if its internal grid is configured properly.

Minimum Size for Rows and Columns

Grid also allows you to enforce a minimum size for rows and columns using the `minsize` option.

```
root.columnconfigure(0, weight=1, minsize=200)
```

This ensures that the column never shrinks below 200 pixels, even when the window is resized smaller.

Disabling Window Resizing

In some cases, you may want to prevent the user from resizing the window entirely. This can be done by disabling resizing on the root window.

```
root.resizable(False, False)
```

This locks both horizontal and vertical resizing.