

# Overview

---

By default, widgets placed using the `grid` geometry manager occupy a single cell. Many real user interfaces, however, require certain widgets—such as titles, wide input fields, or large buttons—to take up more space than a single row or column.

This lesson explains how widgets can span multiple grid cells using `columnspan` and `rowspan`, and how this idea closely matches table-based layouts you may already be familiar with.

## Important Information

---

### Spanning Multiple Cells

When placing a widget with `grid`, you can tell it to occupy more than one cell by specifying how many rows or columns it should cover. This is done using the `columnspan` and `rowspan` options.

The `columnspan` option controls how many columns a widget stretches across, while `rowspan` controls how many rows it stretches across. These options work in a very similar way to the `colspan` and `rowspan` attributes used in HTML tables.

```
label = ttk.Label(root, text="Title")
label.grid(row=0, column=0, columnspan=2)
```

In this example, the label begins in column `0` and stretches across two columns. Grid treats both of those columns as occupied by the label, so other widgets in the same row must be placed outside that span.

### Column Spanning

Column spanning allows a widget to stretch horizontally across multiple columns. This is commonly used when a widget needs more horizontal space than a single column provides.

```
entry = ttk.Entry(root)
entry.grid(row=1, column=0, columnspan=3)
```

Wide text entry fields, section headers, and buttons that should align with several columns of content are all good candidates for column spanning. Using `columnspan` helps maintain alignment while keeping the layout simple and readable.

### Row Spanning

Row spanning allows a widget to stretch vertically across multiple rows.

```
image_label = ttk.Label(root, text="Image")
image_label.grid(row=1, column=0, rowspan=2)
```

This technique is often used for icons, side panels, or visual elements that relate to more than one row of content. Row spanning can help group related information without duplicating widgets.

### Combining Columnspan and Rowspan

A widget can span both rows and columns at the same time.

```
frame = ttk.Frame(root)
frame.grid(row=0, column=0, rowspan=2, columnspan=2)
```

This approach is useful when placing a frame that contains its own internal layout. The frame acts as a larger cell that can hold a more complex arrangement of widgets.

### Example Layout with Spanning Widgets

The following example demonstrates a small interface where some widgets span multiple cells.

```
from tkinter import *
from tkinter import ttk

root = Tk()
root.title("Grid Span Example")

# Title spanning two columns
title = ttk.Label(root, text="User Information")
title.grid(row=0, column=0, columnspan=2)

# Labels and entries
name_label = ttk.Label(root, text="Name:")
name_label.grid(row=1, column=0)

name_entry = ttk.Entry(root)
name_entry.grid(row=1, column=1)

email_label = ttk.Label(root, text="Email:")
email_label.grid(row=2, column=0)

email_entry = ttk.Entry(root)
email_entry.grid(row=2, column=1)

# Button spanning two columns
submit_button = ttk.Button(root, text="Submit")
submit_button.grid(row=3, column=0, columnspan=2)

root.mainloop()
```

In this layout, the title spans across two columns at the top, the form fields are aligned neatly in rows beneath it, and the submit button stretches across the full width of the form. The result is a clean and well-structured interface without relying on exact pixel placement.

### How Grid Handles Spanned Cells

When a widget spans multiple cells, grid treats the spanned area as a single logical space. The cells covered by the span are considered occupied, and other widgets must be placed outside that range. Grid also adjusts row heights and column widths automatically based on the size of the widgets involved.

Grid does not visually merge cells. Instead, it uses the span information to manage layout behind the scenes.

### Designing with Spans in Mind

When planning layouts that use `rowspan` and `columnspan`, it helps to sketch the grid structure ahead of time. Identifying which elements need extra space early on makes the layout easier to build and modify later. Leaving gaps in row or column numbering can also make future changes simpler.

Used thoughtfully, spanning allows you to create clear, professional interfaces while keeping your layout logic straightforward and flexible.