Visual Programming

College Of Computer Sciences and Mathematics

Dr. Firas Aswad

Dept. of Networks

Feb. 19, 2025

The Tkinter geometry manager

Three geometry managers in Tkinter let you specify the position of widgets inside a top-level or parent window.

1. **Pack**: It is simple for simpler layouts but may get very complex for slightly complex layouts.
2. **Grid**: This is the most commonly used geometry manager and provides a table-like layout of management features for easy layout management.
3. **Place**: This is the least popular, but it provides the best control for the absolute positioning of widgets.

The pack geometry manager

For the Pack, imagine the root as an elastic sheet with a small opening at the center. The pack geometry manager makes a hole in the elastic sheet that is just large enough to hold the widget.

The widget is placed along a given inner edge of the gap. It then repeats the process till all the widgets are accommodated.

Finally, when all the widgets have been packed in the elastic sheet, the geometry manager calculates the bounding box for all the widgets. It then makes the parent widget large enough to hold all the child widgets.

When packing child widgets, the pack manager distinguishes between the following three kinds of space:

- Unclaimed space
- Claimed but unused space
- Claimed and used space

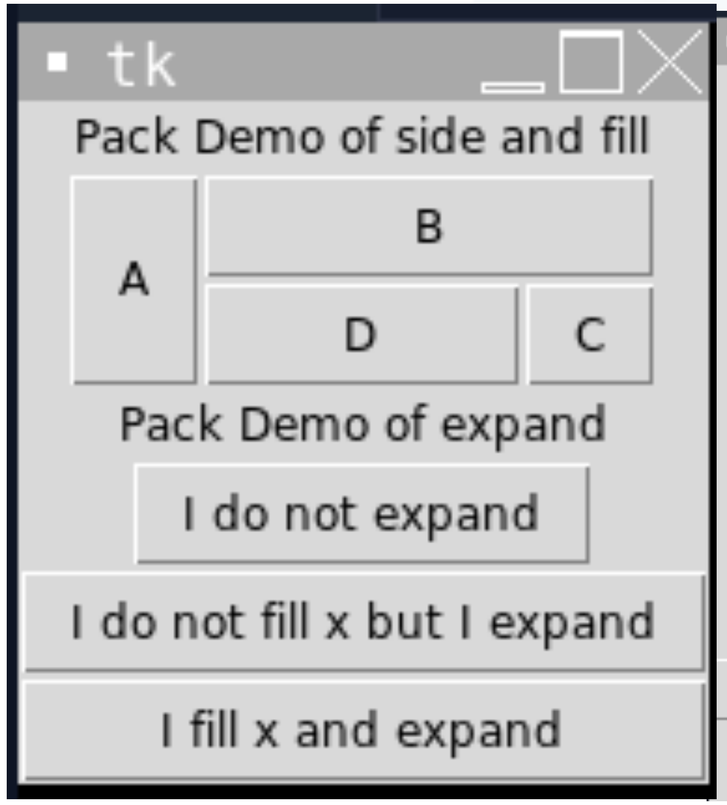
Most Common Pack Options

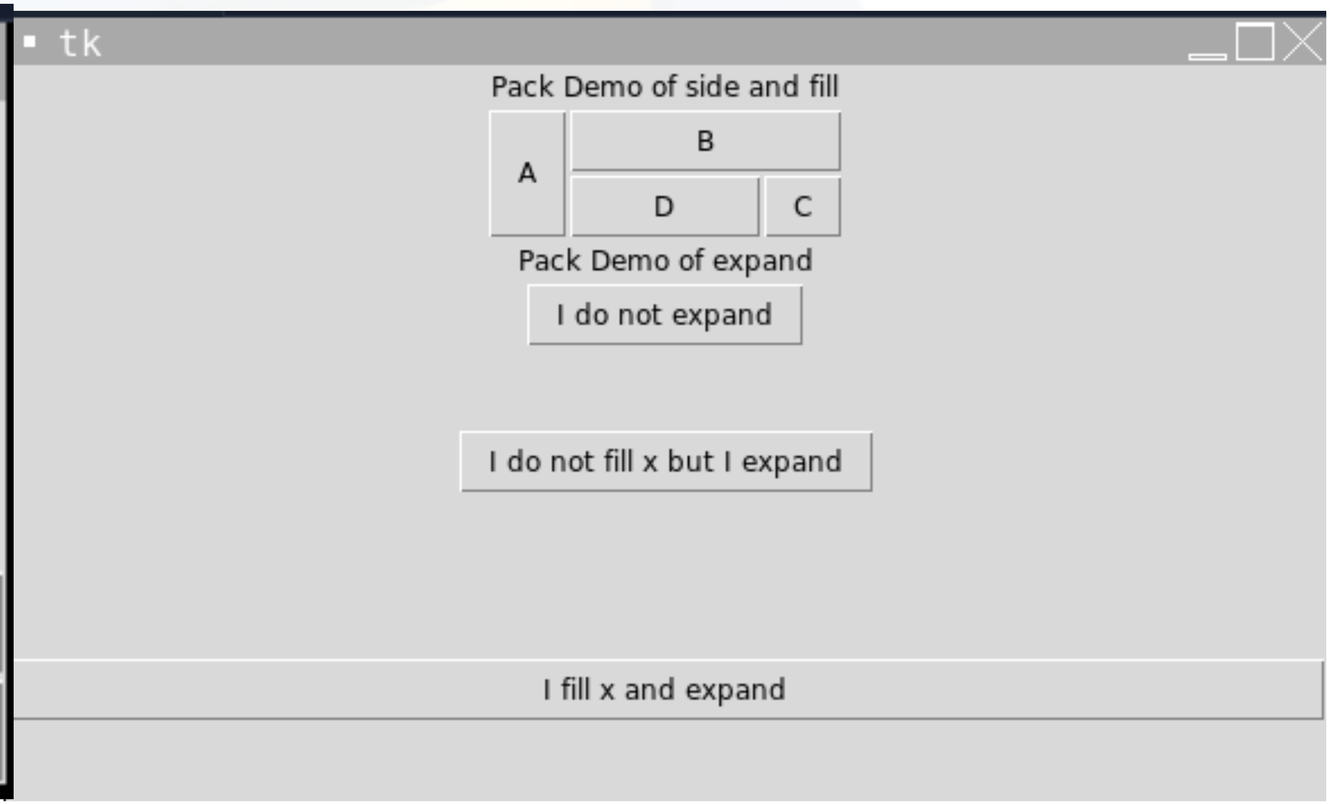
The most commonly used options in pack include the following:

- **side:** LEFT, TOP, RIGHT, and BOTTOM (these decide the alignment of the widget)
- **fill:** X, Y, BOTH, and NONE (these decide whether the widget can grow in size)
- **expand:** Boolean values such as tkinter.YES/tkinter.NO, 1 / 0, and True/False
- **anchor:** NW, N, NE, E, SE, S, SW, W, and CENTER (corresponding to the cardinal directions)
- Internal padding (ipadx and ipady) for the padding inside widgets and external padding (padx and pady), which all default to a value of zero.

The pack geometry manager

Home Work





The pack geometry manager

```
tk.Label(frame, text="Pack Demo of side and fill").pack()
tk.Button(frame, text="A").pack(side=tk.LEFT, fill=tk.Y)
tk.Button(frame, text="B").pack(side=tk.TOP, fill=tk.X)
tk.Button(frame, text="C").pack(side=tk.RIGHT, fill=tk.NONE)
tk.Button(frame, text="D").pack(side=tk.TOP, fill=tk.BOTH)
```

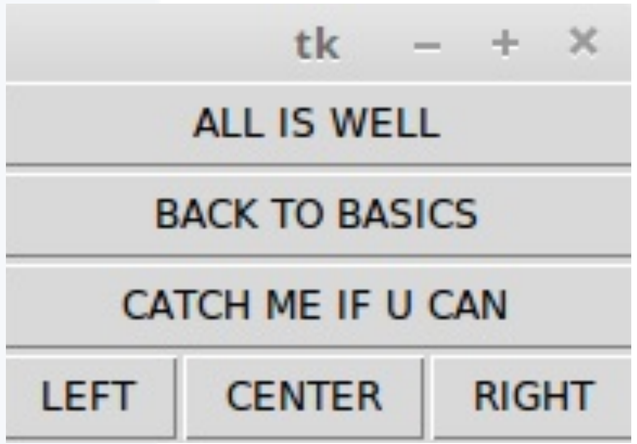
```
tk.Label(root, text="Pack Demo of expand").pack()
tk.Button(root, text="I do not expand").pack()
tk.Button(root, text="I do not fill x but I expand").pack(expand=1)
tk.Button(root, text="I fill x and expand").pack(fill=tk.X, expand=1)
```

The Pack Manager

The pack manager is ideally suited for the following two kinds of situations:

- Placing widgets in a top-down manner

```
tk.Button(parent, text='ALL IS WELL').pack(fill=tk.X)
tk.Button(parent, text='BACK TO BASICS').pack(fill=tk.X)
```



- placing widgets side by side

```
tk.Button(parent, text='LEFT').pack(side=tk.LEFT)
tk.Button(parent, text='CENTER').pack(side=tk.LEFT)
tk.Button(parent, text='RIGHT').pack(side=tk.RIGHT)
```

Pack: Complicated but Great!

Using the pack manager is somewhat complicated compared to the grid method, but it is a great choice for the following situations:

- Having a widget fill the complete container frame
- Placing several widgets on top of each other or side by side.

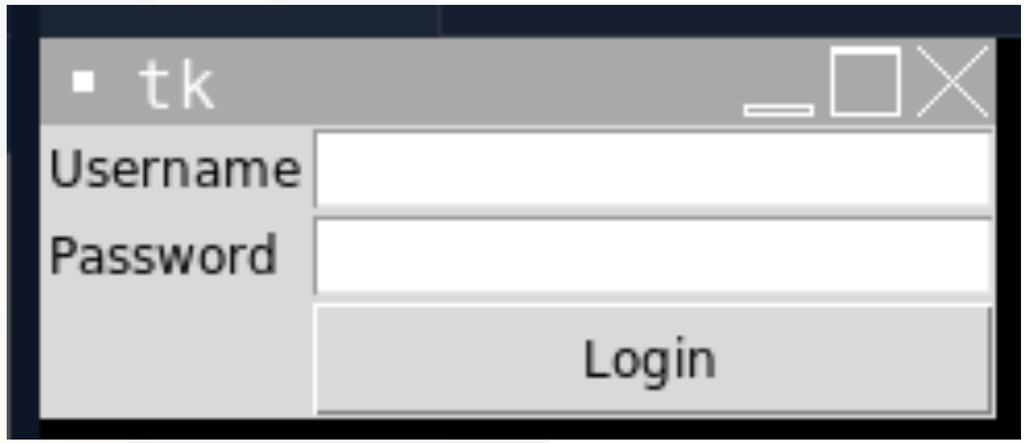
The Grid Geometry Manager

The grid geometry manager is easy to understand. The central idea of the grid geometry manager is to organize the container frame into a two-dimensional table divided into many rows and columns.

In the grid method,

- Each cell can hold only one widget. However, widgets can be made to span multiple cells.
- It is possible to align the widget's position within each cell using the sticky option.
- The sticky option decides how the widget is expanded.
- If the container cell is larger than the size of the widget that it contains, the sticky option can be specified using one or more of the N, S, E, and W options or the NW, NE, SW, and SE options.

Home Work



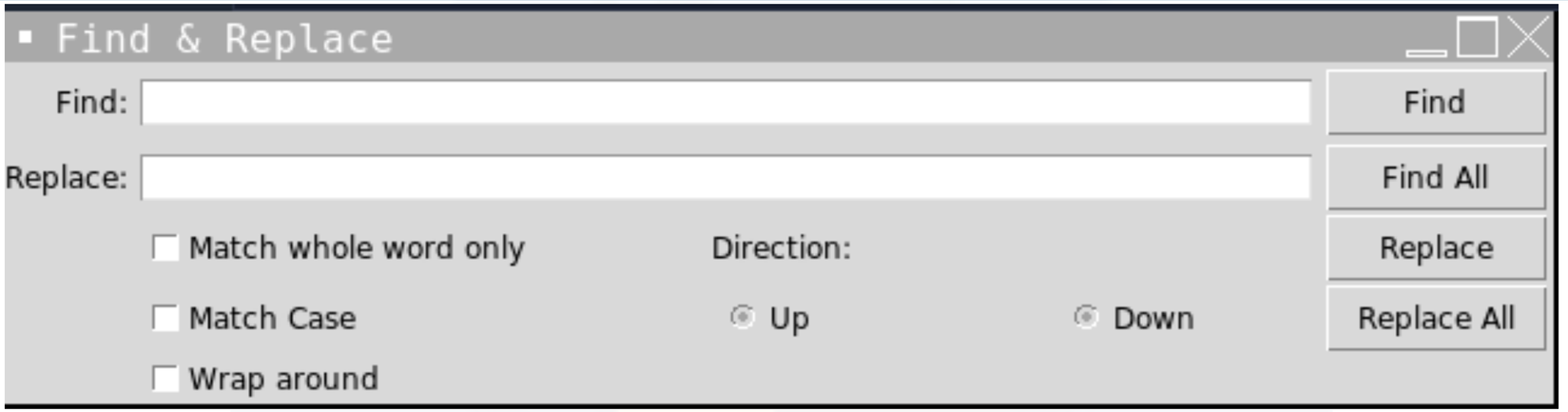

```
tk.Label(root, text="Username").grid(row=0, sticky=tk.W)
tk.Label(root, text="Password").grid(row=1, sticky=tk.W)
tk.Entry(root).grid(row=0, column=1, sticky=tk.E)
tk.Entry(root).grid(row=1, column=1, sticky=tk.E)
tk.Button(root, text="Login").grid(row=2, column=1, sticky=tk.E)
```

Row and Column Span and Pads

- In a more complex scenario, your widgets may span multiple grid cells. The grid method offers handy options such as **rowspan** and **columnspan** to make a grid span multiple cells.

- The grid manager provides the **padx** and **pady** options to provide padding that needs to be placed around a widget.

- Similarly, the **ipadx** and **ipady** options are used for internal padding. These options add padding within the widget itself. The default value of external and internal padding is **0**.




```
tk.Label(parent, text="Find:").grid(row=0, column=0, sticky='e')
tk.Entry(parent, width=60).grid(row=0, column=1, padx=2, pady=2, sticky='we', columnspan=9)

tk.Label(parent, text="Replace:").grid(row=1, column=0, sticky='e')
tk.Entry(parent).grid(row=1, column=1, padx=2, pady=2, sticky='we', columnspan=9)

tk.Button(parent, text="Find").grid(row=0, column=10, sticky='e'+'w', padx=2, pady=2)
tk.Button(parent, text="Find All").grid(row=1, column=10, sticky='e'+'w', padx=2)
```

Forget and Remove Grid

1 – The widget.grid_forget() method can be used to hide a widget from the screen.

What it does: This method removes the widget from the grid layout entirely, as if it was never gridded.

Effect on-grid options: It "forgets" all the grid-related information (e.g., row, column, sticky, padding) associated with the widget. When you later re-grid the widget, you need to specify these options again.

Use case: Use this when you want to completely remove a widget and don't care about preserving its previous grid configuration.

Forget and Remove Grid

2- The **widget.grid_remove()** method removes the widget, except when you make the widget visible again, all of its grid options will be restored.

What it does: This method also hides the widget from the grid layout, but unlike grid_forget(), it retains all the grid-related information (e.g., row, column, sticky, padding).

Effect on-grid options: The widget is hidden, but its grid configuration is preserved. When you later call grid() on the widget, it will reappear in the same place with the same grid options.

Use case: Use this when you want to temporarily hide a widget but plan to show it again in the same position with the same grid settings.

Why Grid?

Where should the grid geometry manager be used?

- The grid manager is a great tool for the development of complex layouts. Complex structures can be easily achieved by breaking the container widget into grids of rows and columns and then placing the widgets in grids where they are wanted.

- It is also commonly used to develop different kinds of dialog boxes.

Grid's column and row sizes

- Different widgets have different heights and widths. So, when you specify the position of a widget in terms of rows and columns, the cell automatically expands to accommodate the widget.

- Normally, the height of all the grid rows is automatically adjusted so it's the height of its tallest cell. Similarly, the width of all the grid columns is adjusted so it's equal to the width of the widest widget cell.

- If you want a smaller widget to fill a larger cell or stay on any one side of the cell, you can use the sticky attribute on the widget to control this aspect.

To override automatic sizing of columns and rows by using the following code:

- w.columnconfigure(n, option=value, ...) AND

- w.rowconfigure(n, option=value, ...)

Minisize, Pad, and Weight Options

Use these to configure the options for a given widget, w, in either the nth column or the nth row, specifying values for the options, minsize, pad, and weight.

Note that the numbering of rows begins from 0 and not 1.

The options available are as follows:

- minsize

This is the minimum size of a column or row in pixels. If there is no widget in a given column or row, the cell does not appear in spite of this minsize specification.

- pad

This is the external padding in pixels that will be pad added to the specified column or row over the size of the largest cell.

- weight

This specifies the relative weight of a row or column and then distributes the extra space. This enables making the row or column stretchable. For example, the following code distributes two-fifths of the extra space to the first column and three-fifths to the second column:

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

The columnconfigure() and rowconfigure() methods are often used to implement the dynamic resizing of widgets, especially on resizing the root window.

- Note: You cannot use the grid and pack methods together in the same container window. If you try doing that, your program will raise a _tkinter.TclError error.

Thank You!