# Overview

A `frame` is a widget that displays as a simple rectangle. Frames help to organize your user interface, often both visually and at the coding level. Frames often act as master widgets for a geometry manager like grid, which manages the slave widgets contained within the frame.

# Important Information

## Creating a Frame

Frames are created using the `ttk.Frame` class:

```
# 'parent' represents whatever container you want to put it in. This can
be the root, or another frame.
frame = ttk.Frame(parent)
```

Frames can take several different configuration options, which can alter how they are displayed.

## Requested size

Typically, the size of a frame is determined by the size and layout of any widgets within it. In turn, this is controlled by the geometry manager that manages the contents of the frame itself.

If, for some reason, you want an empty frame that does not contain other widgets, you can instead explicitly set its size using the `width` and/or `height` configuration options (otherwise, you'll end up with a very small frame indeed).

```
frame['width'] = 500
frame['height'] = 400
```

### Oh No! My Frame Isn't Displaying With It's Intended Size!

By default, frames shrink to fit their internal content! If you put another element inside of a frame, the frame will shrink! You might be wondering "What happened to my frame?".

To prevent your frame from shrinking when adding elements, make sure that it has it's propgate set to false.

```
# Add the frame to the display.
frame.grid()

# Prevent the frame from shrinking.
frame.grid_propagate(False)
```

## Specifying Units

When we asked that the frame be "500" wide, how big is that? What are the units? If not otherwise specified, screen distances such as width and height (and others, like padding and borderwidth below) are measured in pixels. So when we ask that a frame be "500" wide, we're requesting 500 pixels.

You can also specify a number of different units for screen distances by appending one of several suffixes to the number. For example, `350` means 350 pixels, `350c` means 350 centimeters, `350m` means 350 millimeters, `350i` means 350 inches, and `350p` means 350 printer's points (1/72 inch). While if we're specifying pixels we may have passed the screen distance as an integer, when we use one of these different units, we're passing the screen distance as a string.

```
frame['width'] = '500p'    # 500 points (just under 7 inches)
```

## Padding

The `padding` configuration option is used to request extra space around the inside of the widget. If you're putting other widgets inside the frame, that provides a margin between the frame's border and the widgets within. You can specify the same padding for all sides, different horizontal and vertical padding, or padding for each side separately.

```
# 5 pixels on all sides
frame['padding'] = 5

# 5 on left and right, 10 on top and bottom
frame['padding'] = (5, 10)

# left: 5, top: 7, right: 10, bottom: 12
frame['padding'] = (5, 7, 10, 12)
```

## Changing Styles

Frames have a `style` configuration option, **which is common to all of the themed widgets**. This lets you control many other aspects of their appearance or behavior. This is a bit more advanced, so we won't go into it in too much detail right now. But here's a quick example of creating a "Danger" frame with a red background and a raised border.

```
s = ttk.Style()
s.configure('Danger.TFrame', background='red', borderwidth=5,
relief='raised')
frame = ttk.Frame(root, width=200, height=200, style='Danger.TFrame')
```

## Adding The Widget to the Window

Before we focus on creating layouts with geometry later in the lessons on tkinter, we can add wigets to the window with `.pack()`. This method will stack widgets on top of one another vertically in the center of the window, and works for the purpose of displaying elements while we are still learning how to create widgets.

```
frame.pack()
```

We can also use `.grid()` to add widgets to our window. When we learn about geometry, we will learn about using `.grid()` to create more intentional layouts. While we are still learning about creating widgets. Adding elements with `.grid()` with no additional configuration will stack them vertically on the left side of the window.

```
frame.grid()
```

# Copy, Change, Challenge

## Copy

```
from tkinter import *
from tkinter import ttk

root = Tk()
root.title("Frame Copy Example")

frame = ttk.Frame(root)

frame['width'] = 300
frame['height'] = 200
frame['padding'] = 10

# Add a style to make the frame visible
s = ttk.Style()
s.configure('Cool.TFrame', background='blue', borderwidth=5,
relief='raised')

frame['style'] = 'Cool.TFrame'

# Add the frame to the window
frame.grid()

root.mainloop()
```

## Change

Modify the program so that:

- The frame is **500 pixels wide** and **400 pixels tall**
- The padding is **5 pixels on the left and right** and **15 pixels on the top and bottom**
- The frame is added to the window using `.pack()` **instead of** `.grid()`

## Challenge

Create a program with **two frames**:

- One frame should:
    - Use pixel values for width and height
    - Have equal padding on all sides
- The other frame should:
    - Use a non-pixel unit (such as points) for its width
    - Use different padding values for each side
- Place both frames in the window so they are clearly visible