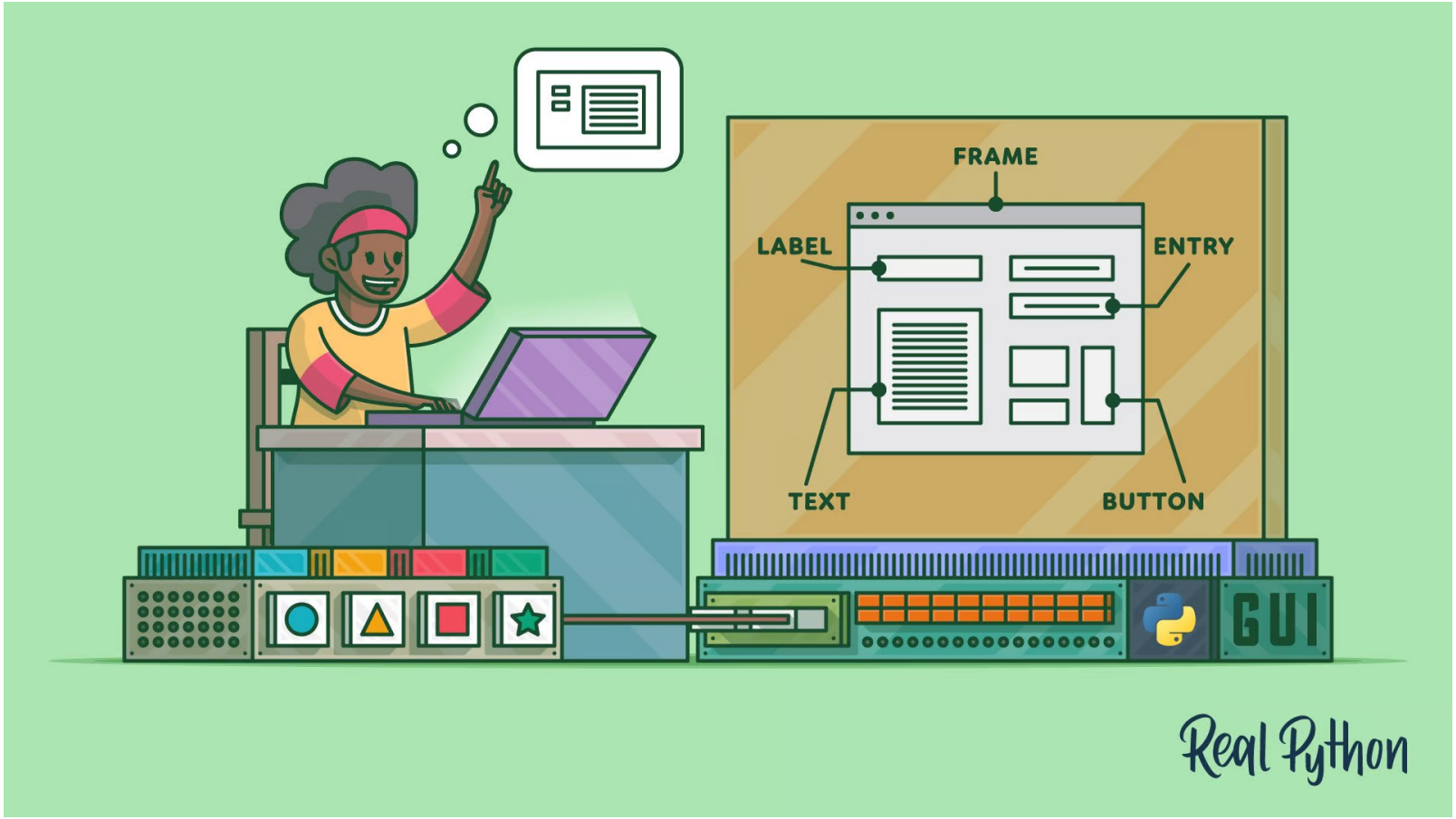
Building a Python GUI Application With `tkinter`



Building a Python GUI Application With `tkinter`

Building a Python GUI Application With `tkinter`

- Get Started With `tkinter`
- Work With Widgets
- Control Application Layout
- Make Applications Interactive
- Build Two Applications:
  - A Temperature Converter
  - A Text Editor

Python Version

- Created With Python 3.12
- All Code Should Work in Recent Python Versions

Let’s Get Started!

Building a Python GUI Application With `tkinter`

1. **Getting `tkinter` Up and Running**
2. Building Your First GUI Application With `tkinter`
3. Working With Widgets
4. Controlling Layout With Geometry Managers
5. Making Your Applications Interactive
6. Building A Temperature Converter
7. Building A Text Editor

Getting `tkinter` Up and Running

- Installed from Official Installers:
  - Windows
  - macOS
  - `tkinter` Present - ✅
  - `import tkinter as tk` Should Not Generate an Error
- Installed from:
  - Other Sources
  - No Official Distribution Available
  - `tkinter` Installation May Not Be Present

Python on macOS with `homebrew`

- Does Not Include Tcl/Tk Dependency
- Default System Version Used
- May be Outdated and `tkinter` May Not Work
- Use Installer from `www.python.org`

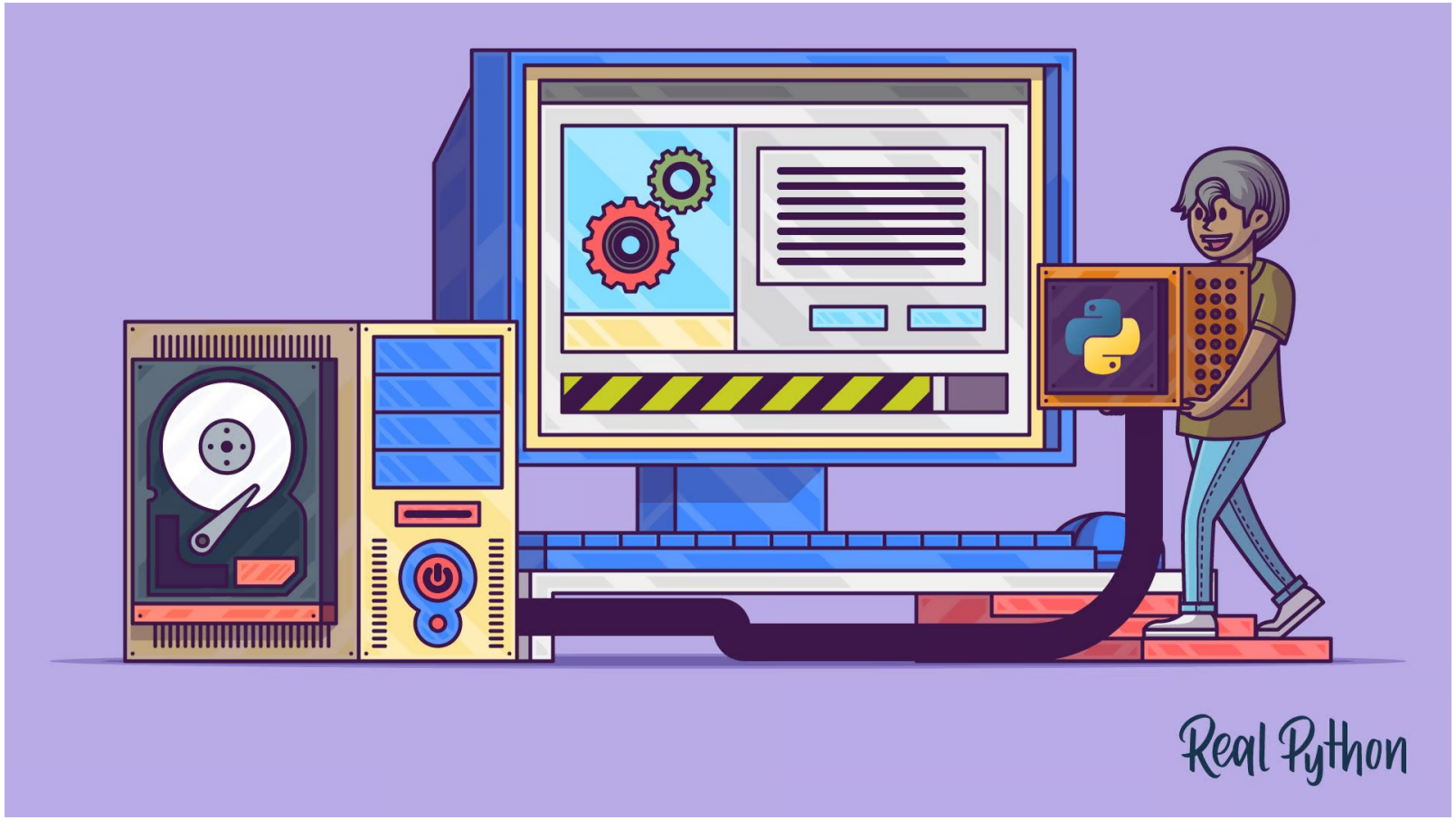
Ubuntu Linux 22.04

- Default Python has no `tkinter` Support
- Install Via `sudo apt-get install python3-tk`

Other Linux Distributions

- Build Python from Source Code

Python 3 Installation & Setup Guide



https://realpython.com/installing-python/#compiling-python-from-source

Alternate Python Shells

- Using:
  - `Bpython`
  - `ptpython`
- `tkinter` Window May Not Appear Instantly
- Standard Python REPL Will React as Seen on Screen

Next: Building Your First GUI Application With `tkinter`

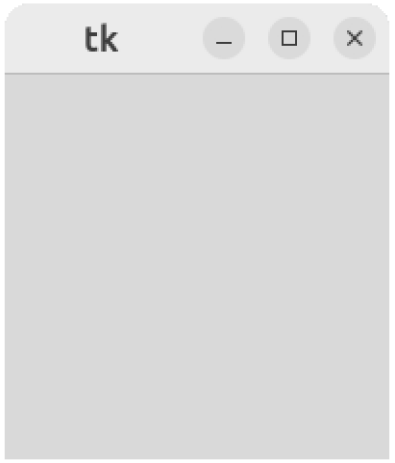
Building a Python GUI Application With `tkinter`

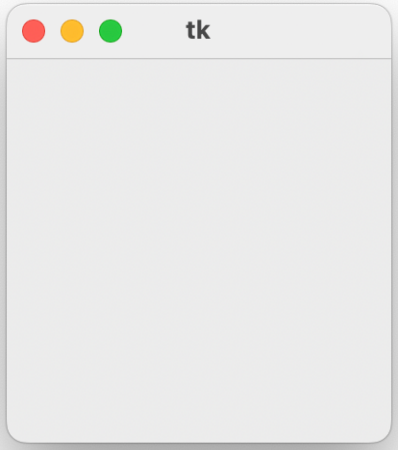
1. Getting `tkinter` Up and Running
2. **Building Your First GUI Application With `tkinter`**
3. Working With Widgets
4. Controlling Layout With Geometry Managers
5. Making Your Applications Interactive
6. Building A Temperature Converter
7. Building A Text Editor

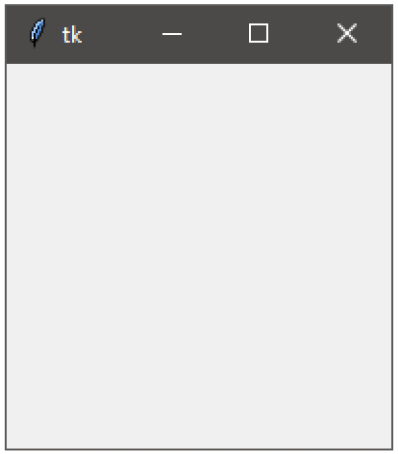
The Window

- Foundational Element of GUI Programming
- Container for All Other Elements:
  - Text Boxes
  - Labels
  - Buttons
- Elements are Known as Widgets

The Same Window on Linux, macOS and Windows







Working With `tkinter`

- In the REPL:
  - Updates Are Applied As Each Line Is Executed
- In a Python Script:
  - `window.mainloop()` Is Needed:
    - Application Will Not Run if Omitted

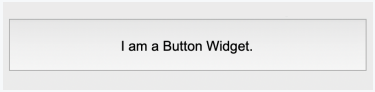
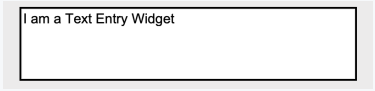
Next: Working With Widgets

Building a Python GUI Application With `tkinter`

Widgets

- The Bread and Butter of `tkinter`
- Provide Elements for User Interaction
- Each Widget is Defined by a Class

`tkinter` Example Widgets

Class	Image	Description
`Label`		A Widget Used To Display Text on the Screen
`Button`		A Button That Can Contain Text and Can Perform an Action When Clicked
`Entry`		A Text Entry Widget That Allows Only a Single Line of Text
`Text`		A Text Entry Widget That Allows Multiline Text Entry
`Frame`		A Rectangular Region Used To Group Related Widgets or Provide Padding Between Widgets

TkDocs: Basic Widgets

TkDocs Tutorial Resources Book About Search TkDocs

This tutorial will quickly get you up and running with the latest Tk from Python, Tcl, Ruby, and Perl on macOS, Windows, or Linux. It provides all the essentials about core Tk concepts, the various widgets, layout, events and more that you need for your application.

Previous: Tk Concepts | Contents | Single Page | Next: The Grid Geometry Manager

Basic Widgets

This chapter introduces the basic Tk widgets that you'll find in just about any user interface: frames, labels, buttons, checkbuttons, radiobuttons, entries, and comboboxes. By the end, you'll know how to use all the widgets you'd ever need for a typical fill-in-the-form type of user interface.

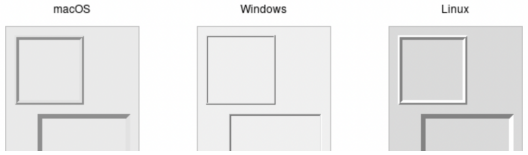
You'll find it easiest to read this chapter (and those following that discuss more widgets) in order. Because there is so much commonality between many widgets, we'll introduce certain concepts when describing one widget that will also apply to a widget we describe later. Rather than going over the same ground multiple times, we'll refer back to when the concept was first introduced.

As each widget is introduced, we'll refer to the widget roundup page for the specific widget, as well as the Tk reference manual page. As a reminder, this tutorial highlights the *most useful parts* of Tk and how to use them to build effective modern user interfaces. The reference documentation, which details *everything* that can be done in Tk, serves a very different purpose.

Frame

- Widget Roundup
- Reference Manual

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.



Essentials

- Tutorial
- Installing Tk
- Modern Tkinter
- Tk Backgrounder
- Official Tk Command Reference
  (Tcl-oriented; at www.tcl.tk)

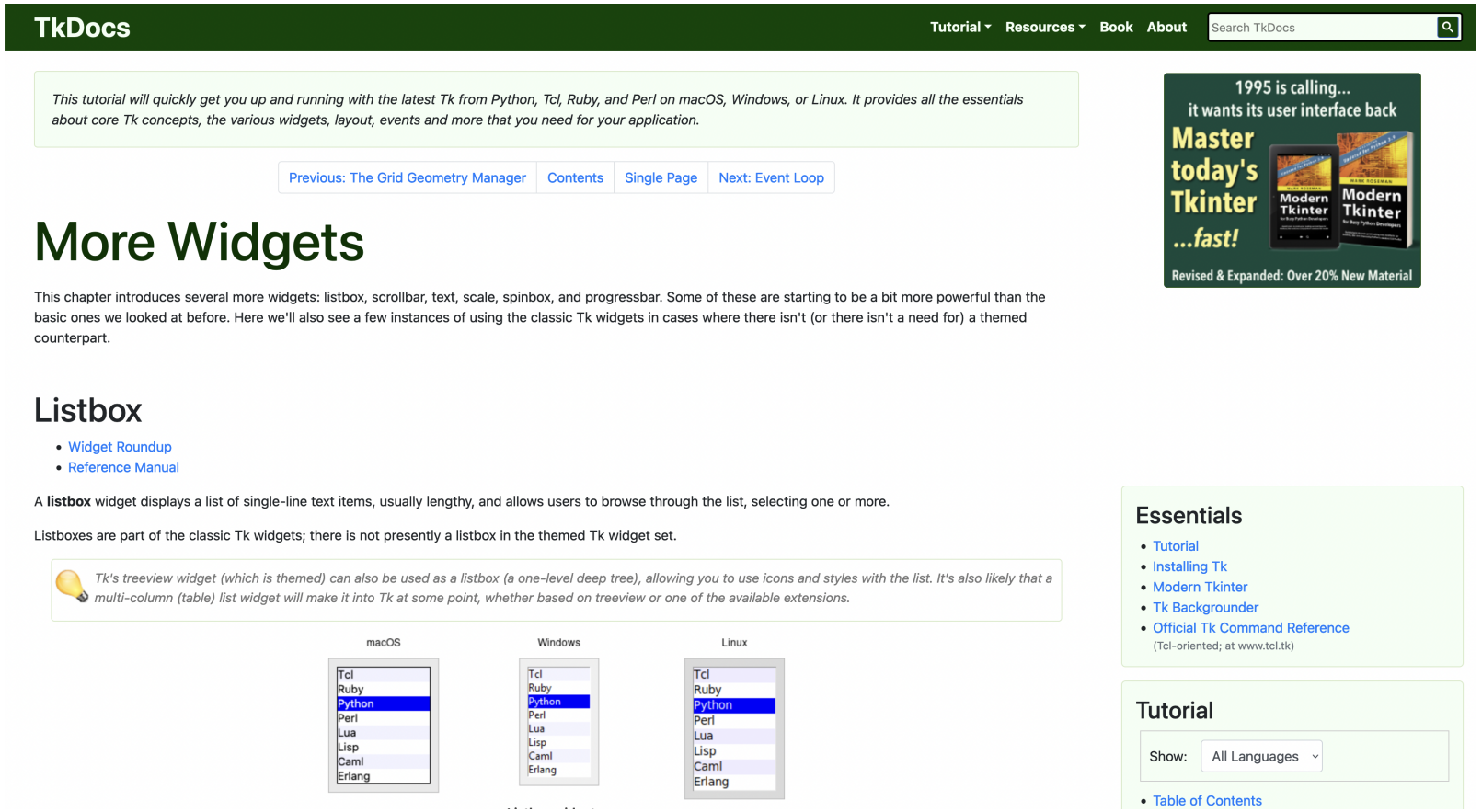
Tutorial

Show: All Languages

- Table of Contents

https://tkdocs.com/tutorial/widgets.html

TkDocs: More Widgets



https://tkdocs.com/tutorial/morewidgets.html

Displaying Text and Images With `Label` Widgets

- Used to Display Text or Images
- Text Is Not User-Editable
- Created by Instantiating `Label` Class:
  - `label = tk.Label(text="Hello, Tkinter")`
- Displayed Using Default System Colors

Tcl/Tk Colors Page

NAME

colors — symbolic color names recognized by Tk

DESCRIPTION

Tk recognizes many symbolic color names (e.g., **red**) when specifying colors. The symbolic names recognized by Tk and their 8-bit-per-channel RGB values are:

Name	Red	Green	Blue
alice blue	240	248	255
AliceBlue	240	248	255
antique white	250	235	215
AntiqueWhite	250	235	215
AntiqueWhite1	255	239	219
AntiqueWhite2	238	223	204
AntiqueWhite3	205	192	176
AntiqueWhite4	139	131	120
agua	0	255	255
aquamarine	127	255	212
aquamarine1	127	255	212
aquamarine2	118	238	198
aquamarine3	102	205	170
aquamarine4	69	139	116
azure	240	255	255
azure1	240	255	255
azure2	224	238	238
azure3	193	205	205
azure4	131	139	139
beige	245	245	220
bisque	255	228	196
bisque1	255	228	196
bisque2	238	213	183
bisque3	205	183	158
bisque4	139	125	107
black	0	0	0
blanched almond	255	235	205
BlanchedAlmond	255	235	205
blue	0	0	255
blue violet	138	43	226
blue1	0	0	255
blue2	0	0	238
blue3	0	0	205
blue4	0	0	139
BlueViolet	138	43	226
brown	165	42	42
brown1	255	64	64

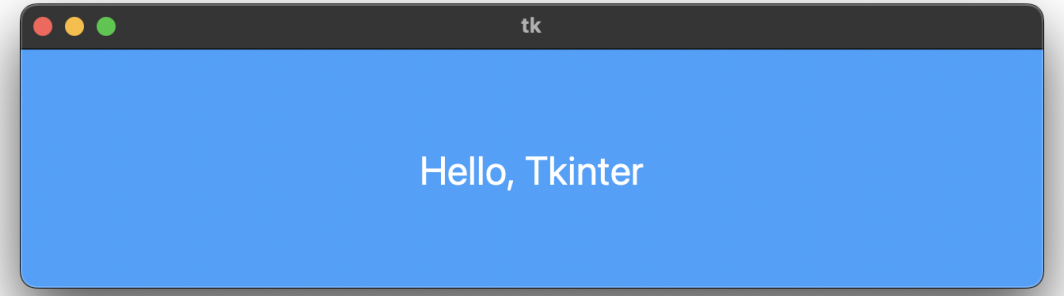
https://www.tcl.tk/man/tcl/TkCmd/colors.html

Color Using Hexadecimal Values

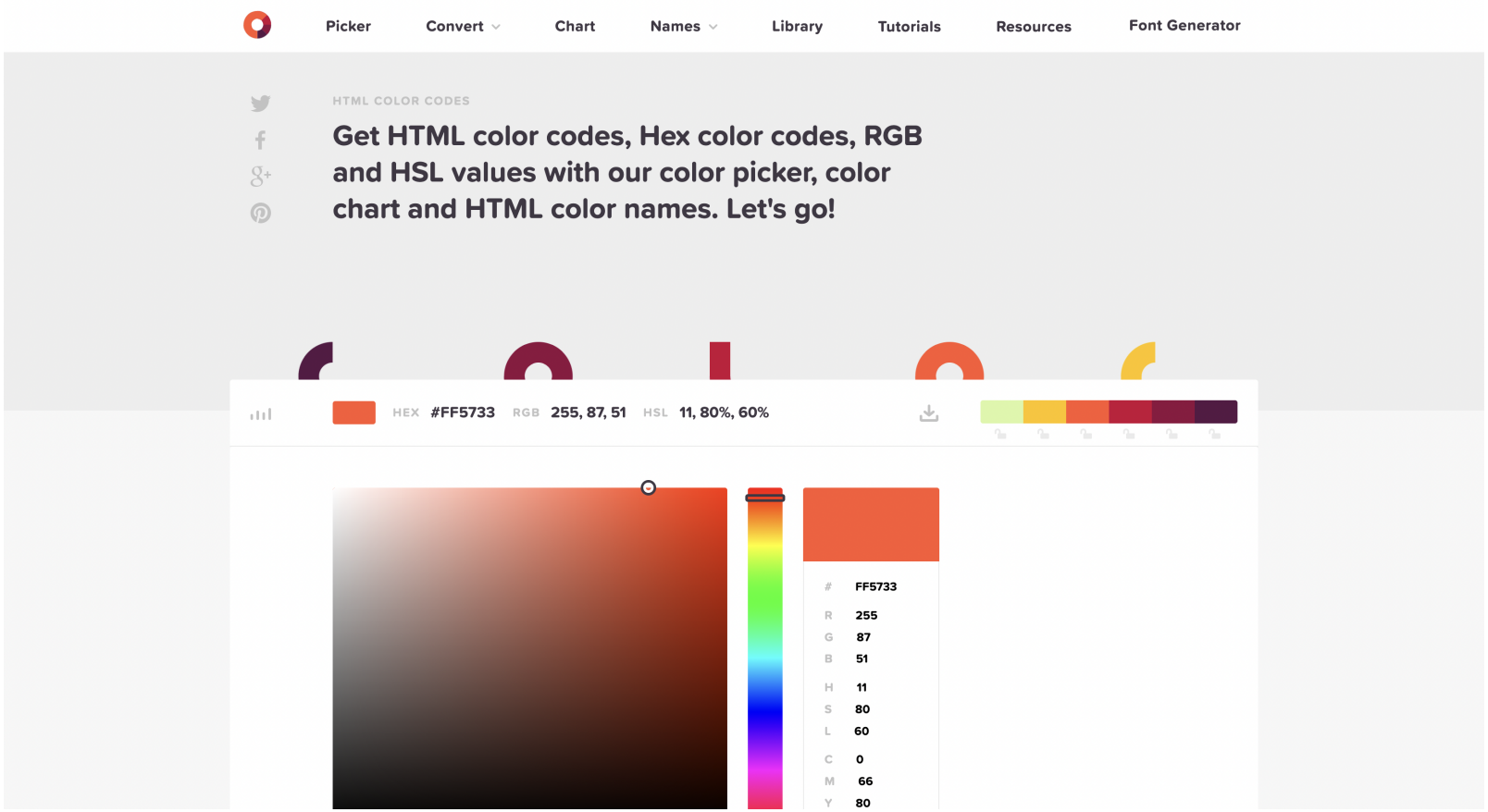
```
label = tk.Label(text="Hello, Tkinter", background="#34A2FE")
```

Color Using Hexadecimal Values

```
label = tk.Label(text="Hello, Tkinter", background="#34A2FE")
```



HTML Color Codes



https://htmlcolorcodes.com/

Shorthand Notation

- `foreground` → `fg`
- `background` → `bg`

Text Units

- Horizontal Unit:
  - Determined by the Width of `0`
- Vertical Unit:
  - Determined by the Height of `0`
- Ensures Consistent Behavior Across Platforms
- Ensures Text Fits in Labels and Buttons

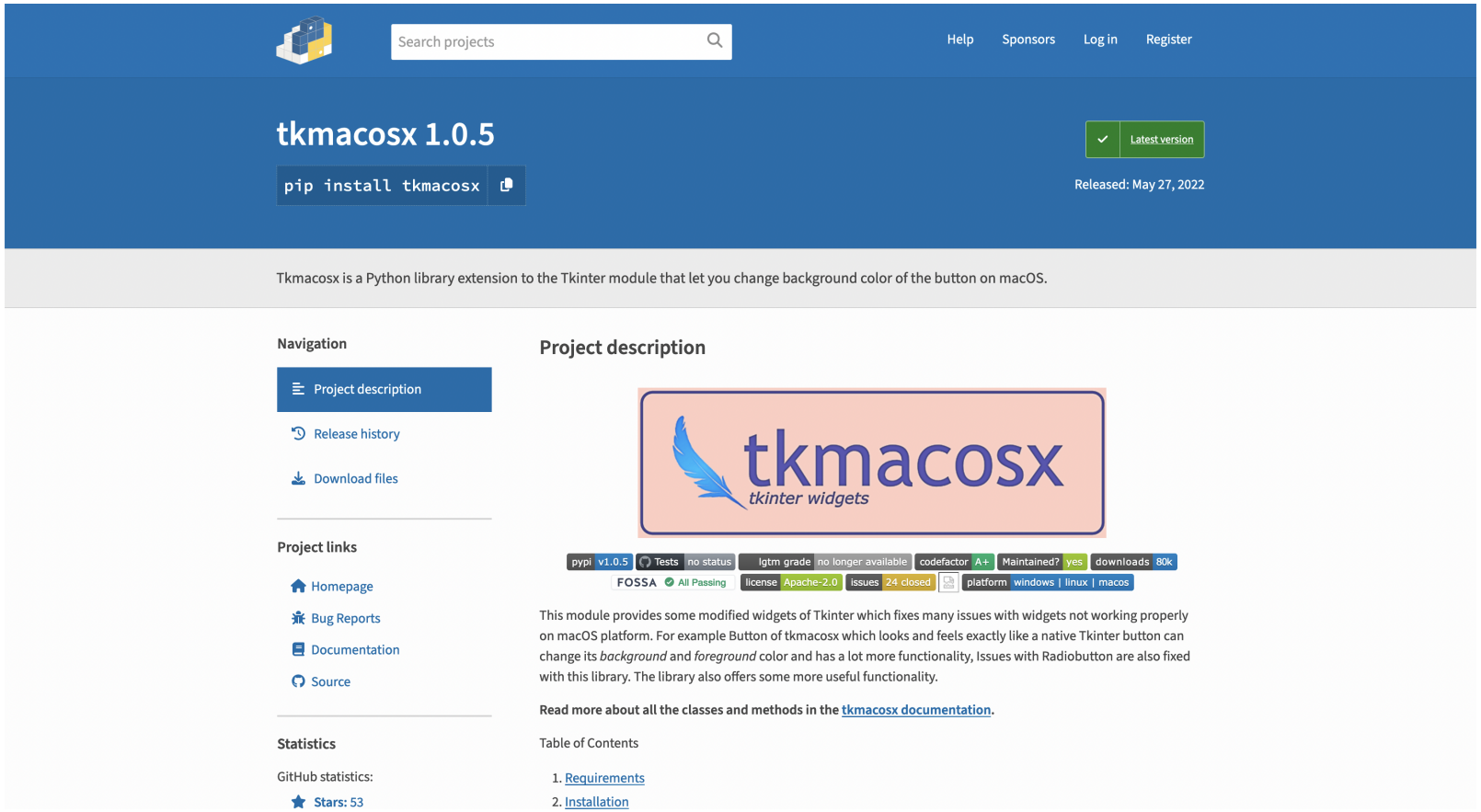
Clickable Buttons With `Button` Widgets

- Used to Display Clickable Buttons
- Can Be Configured to Call A Function When Clicked
- A `Button` is Similar to a `Label`:
  - Adds Click Functionality
  - Style Keyword Arguments Will Work

Button Color on macOS

- `background` Will Not Alter Background Color
- Changes in macOS Limit Control of Button Objects
- An Alternate Solution Is Needed

tkmacosx



https://pypi.org/project/tkmacosx/

Next: Getting User Input With `Entry` Widgets

Building a Python GUI Application With `tkinter`

Getting User Input

- Small Amount of Text From a User:
  - Name
  - Email Address
- Use an `Entry` Widget
- Displays a Small Text Box
- Created and Styled Similarly To `Label` and `Button`

`Entry` Widget Operations

- Retrieving Text: `.get()`
- Deleting Text: `.delete()`
- Inserting Text: `.insert()`

`Entry.delete()`

- Works Like String Slicing:
  - First Argument is Starting Index
  - Deletion Up To But Not Including Second Argument

`Entry.insert()`

- First Argument is Insertion Point
- Second Argument is Text to Insert

Entry Widgets

- Good For Small Amounts of Text
- Unsuitable for Larger Text Inputs

Next: Getting Multiline User Input with `Text` Widgets

Building a Python GUI Application With `tkinter`

`Text` Widgets

- Used for Entering Text
- Can Contain Multiple Lines of Text:
  - Paragraphs
  - Multiple Pages
- Provide Three Main Operations:
  - Retrieving Text: `.get()`
  - Deleting Text: `.delete()`
  - Inserting Text: `.insert()`

`Text.get()`

- **Single Index:** Single Character Returned
- **Start and End Index:** Multiple Characters Returned

Indices in `Text` Widgets

- Work Differently to `Entry` Widgets
- Contain Two Pieces of Information:
  - The Line Number of a Character (1-Indexed)
  - The Position of a Character on That Line (0-Indexed)
- String in Format `"<line>.<char>"`:
  - `"1.0"` - First Character on the First Line
  - `"2.3"` - Fourth Character on the Second Line

`Text.delete()`

- Deletes Characters from `Text` Widgets
- Works as `Entry.delete()` Does
- Two Methods of Use:
  - Single Argument
  - Two Arguments

`Text.delete()`

- Deletes Characters from `Text` Widgets
- Works as `Entry.delete()` Does
- Two Methods of Use:
  - Single Argument: Index of Character to Delete
  - Two Arguments

`Text.insert()`

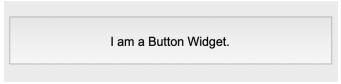
- Insert Text at Specified Position if Text Already Present There
- Append Text to Specified Line if Text Ends Before Insertion Point

Tracking Last Character

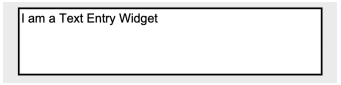
- Usually Impractical
- Pass `tk.END` As Insertion Point

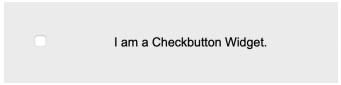
`tkinter` Widgets

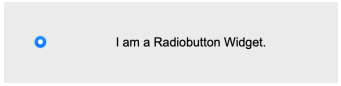
- `Label` - 

- `Button` - 

- `Entry` - 

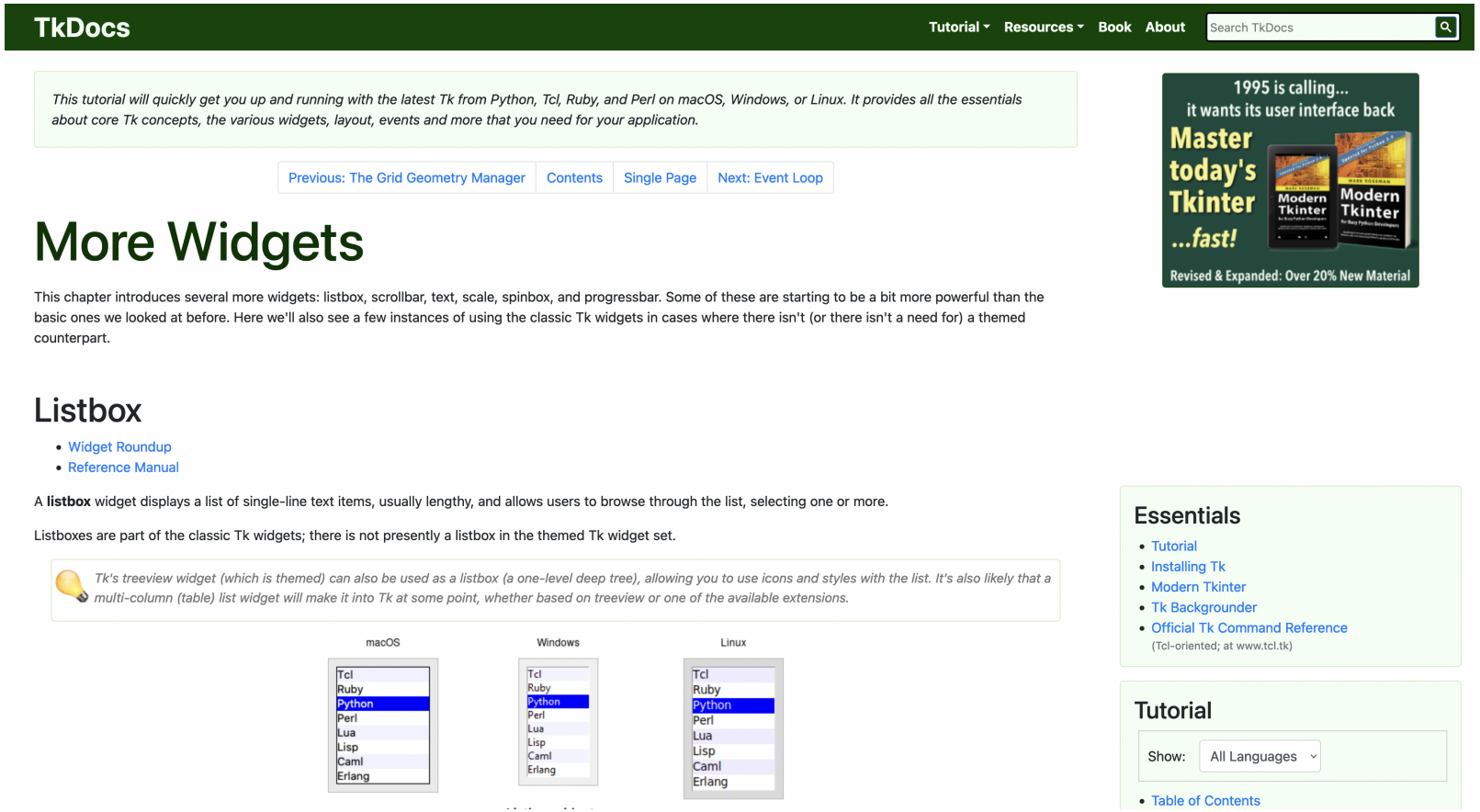
- `Text` - 

- `Checkbutton` - 

- `Radiobutton` - 

- `Scrollbar` -
- `Progressbar` -

TkDocs: More Widgets



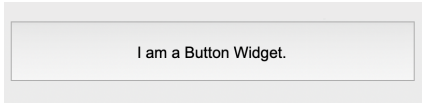
https://tkdocs.com/tutorial/morewidgets.html

Next: Assigning Widgets to Frames with `Frame` Widgets

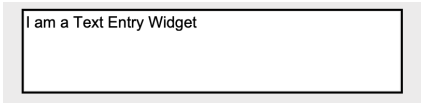
Building a Python GUI Application With `tkinter`

Widgets Used in This Course

- `Label` - 

- `Button` - 

- `Entry` - 

- `Text` - 

- `Frame` -

A Look at `Frame` Widgets

- Empty Frame Widgets Are Practically Invisible
- Containers for Other Widgets
- Widgets Assigned To Frames Via `master` Attribute

Widget `master` Attribute

- Available in `Label`, `Button`, `Entry` and `Text` Widgets
- Set on Widget Instantiation
- Allows Control of Widget Assignment
- Omission Leads to Top-Level Window Placement

`Frame` Widgets

- Used for Organizing Other Widgets
- Related Widgets Assigned to One Frame:
  - Grouped Widgets Stay Together
- Improve Visual Presentation

Next: Adjusting Frame Appearance

Building a Python GUI Application With `tkinter`

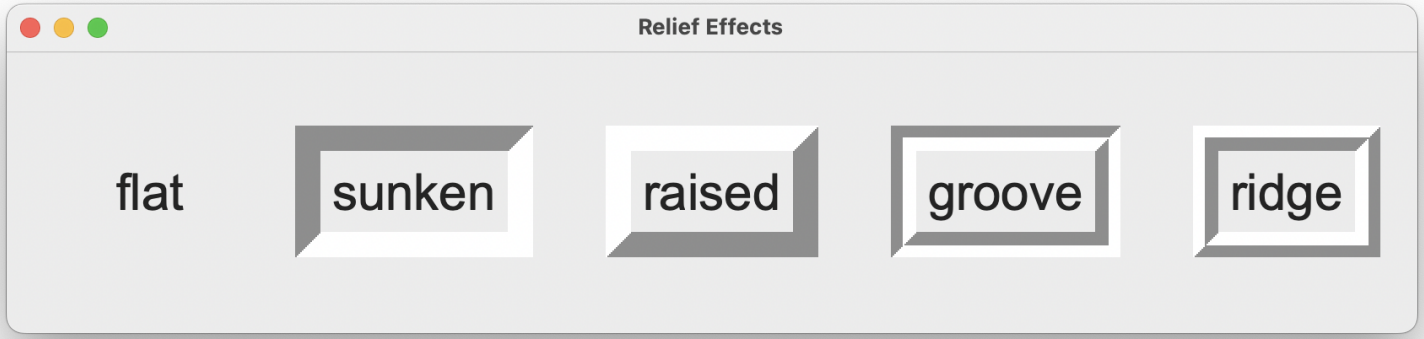
The `relief` Attribute

- `tk.FLAT`: Has No Border Effect (the Default Value)
- `tk.SUNKEN`: Creates a Sunken Effect
- `tk.RAISED`: Creates a Raised Effect
- `tk.GROOVE`: Creates a Grooved Border Effect
- `tk.RIDGE`: Creates a Ridged Effect

The `borderwidth` Attribute

- Must be Greater Than `1` For Effect to Be Visible
- Adjusts Border Width in Pixels

Relief Effects



Understanding Widget Naming Conventions

- Name Must be a Valid Python Identifier
- Including Widget Class is a Good Idea:
  - `Label` Widget: `label_user_name`
  - `Entry` Widget: `entry_age`
  - Helps Anyone Reading Code in Future
  - Full Names Can Lead to Long Variable Names

Shorthand Prefixes for Widget Names

Widget Class	Prefix	Example
`Label`	`lbl`	`lbl_name`
`Button`	`btn`	`btn_submit`
`Entry`	`ent`	`ent_age`
`Text`	`txt`	`txt_notes`
`Frame`	`frm`	`frm_address`

Defining Widgets Without Variable Assignment

```
tk.Label(text="Hello, Tkinter").pack()
```

- Helpful When Widget Will Not Be Referred to Later On
- Unreferenced Objects Would Normally be Garbage Collected
- `tkinter` Avoids This with Internal Registration

Next: Controlling Layout With Geometry Managers

Building a Python GUI Application With `tkinter`

Usage So Far

- Adding Widgets to:
  - Windows
  - `Frame` Widgets
- Using `.pack()`

Geometry Managers

- Control Application Layout
- `.pack()`
- `.place()`
- `.grid()`
- Each Window or `Frame` Uses only One Manager
- Different Managers Can Be In Use Simultaneously

The `.pack()` Geometry Manager

- Uses a Packing Algorithm:
  - Create Rectangular Parcel That Widget Fits Into
  - Center The Widget Unless Otherwise Specified
- Powerful but Can Be Difficult To Visualize
- Accepts Keyword Arguments:
  - `fill` - `tk.X`, `tk.Y`, `tk.BOTH`
  - `side` - `tk.TOP`, `tk.BOTTOM`, `tk.LEFT`, `tk.RIGHT`

Next: The `.place()` and `.grid()` Geometry Managers

Building a Python GUI Application With `tkinter`

The `.place()` Geometry Manager

- Control Widget Location Precisely
- Provide `x` and `y` Keyword Arguments
- Measurements are in Pixels
- Origin (0,0) Is at Top Left
- `y` is Distance from Top of Window
- `x` is Distance from Left of Window

The `.place()` Geometry Manager

- Results Can Vary With Operating Systems and Fonts
- Layout Management Can Be Difficult to Manage
- Layouts are **Not** Responsive
- These Factors Make Cross-Platform Use Difficult
- Can Be Useful When Precision is Paramount
- `.pack()` Is Usually a Better Choice

The `.grid()` Geometry Manager

- Most Commonly Used
- Provides Power of `.pack()`
- Easier to Understand and Maintain
- Splits Area into Rows and Columns
- Location Specified by calling `.grid()`:
  - Pass `row` and `column` Indices
  - Both are Zero-Indexed

Padding And `.grid()`

- Frames Tightly Packed by Default
- Padding is Blank Space
- Padding with `.grid()`:
  - `padx` - Horizontal
  - `pady` - Vertical
  - Set in Pixels

Next: Handling Window Resizing

Building a Python GUI Application With `tkinter`

`.columnconfigure()` and `.rowconfigure()`

- Applied to Window Object
- Adjust Growth as Window is Resized
- Grid is Attached to Window
- Arguments:
  - Index - Column or Row to Configure
  - `weight` - Determines Relative Growth
  - `minsize` - Minimum Size in Pixels

The `weight` Argument

- Set to `0` By Default:
  - No Expansion with Window Resizing
- If All Set to `1`:
  - All Grow At the Same Rate
- One Set to `1`, One Set to `2`:
  - Second Column Expands Twice as Quickly

Label Location

- Set with the `sticky` Parameter:
  - `n` or `N` - Align to the Top-Center of the Cell
  - `e` or `E` - Align to the Right-Center of the Cell
  - `s` or `S` - Align to the Bottom-Center of the Cell
  - `w` or `W` - Align to the Left-Center of the Cell

Widget Positioning with `sticky`

- Widget Only Large Enough to Contain Contents:
  - Will Not Fill Entire Grid Cell
- To Fill Grid:
  - `ns` to Fill Vertically
  - `ew` to Fill Horizontally
  - `nsew` to Fill Entire Cell

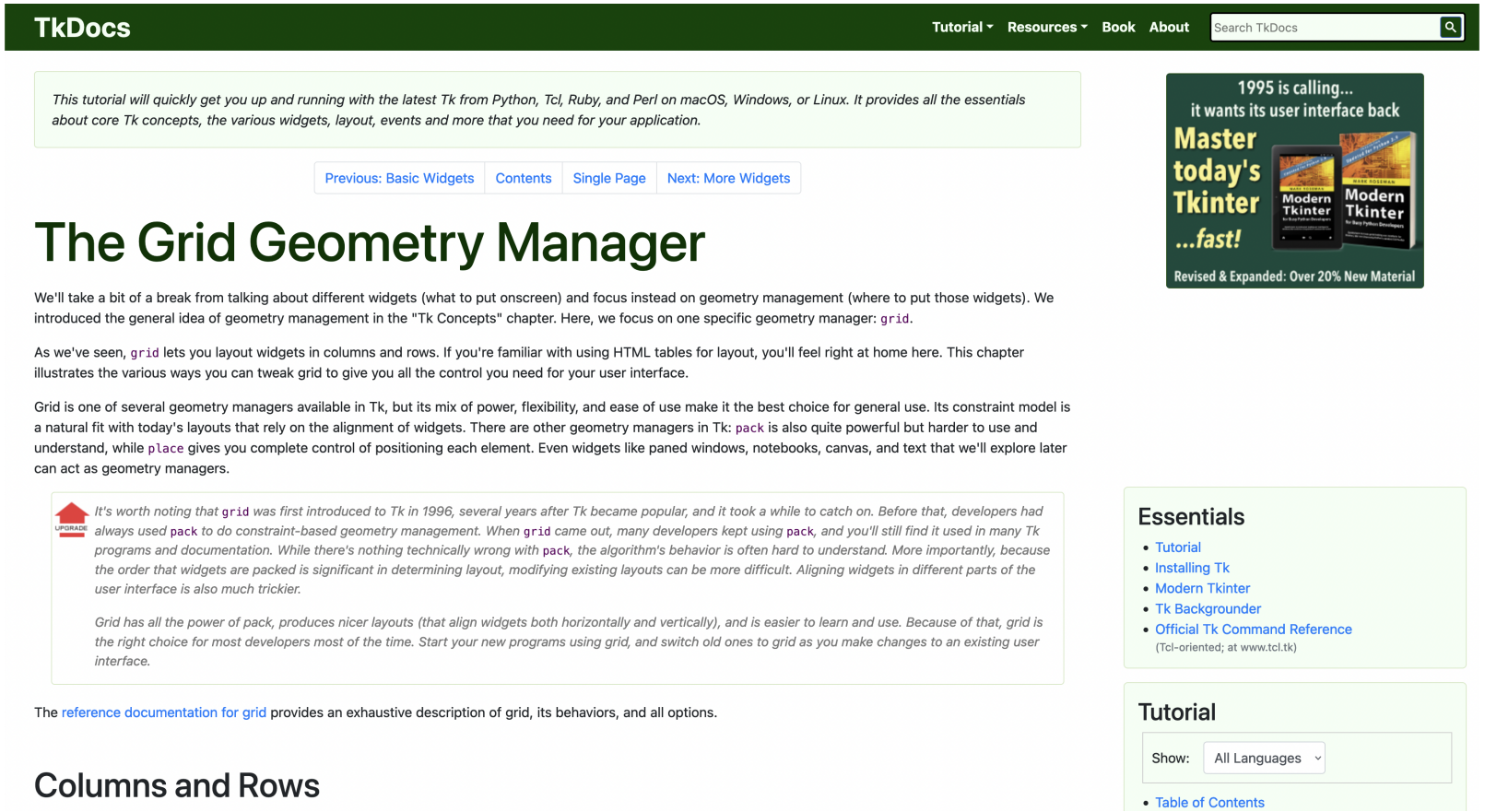
Parallels Between `sticky` and `fill`

.grid()	.pack()
`sticky="ns"`	`fill=tk.Y`
`sticky="ew"`	`fill=tk.X`
`sticky="nsew"`	`fill=tk.BOTH`

The `.grid()` Geometry Manager

- A Powerful Geometry Manager
- Easier to Understand than `.pack()`
- More Flexible than `.place()`
- Often the Best Choice as Primary Geometry Manager
- Offers More Flexibility Than Seen Here

TkDocs: Geometry Manager



https://tkdocs.com/tutorial/grid.html

Next: Making Your Applications Interactive

Building a Python GUI Application With `tkinter`

Progress So Far

- Create a Window - 
- Add Widgets - 
- Control Application Layout - 
- Application Interactivity

Using Events and Event Handlers

- `window.mainloop()` Starts Event Loop:
  - Application Checks for Events
  - Executes Code in Response
- Event Loop Provided by `tkinter`
- Response Code Written by You:
  - Functions Called **Event Handlers**

Events

- Any Action Which Might Trigger Behavior:
  - Key Press
  - Mouse Click
- Event Object Emitted by an Event
  - Instance of Class Representing Event
  - Instantiated by `tkinter`

An Example Event Loop

- Better Understand The `tkinter` Event Loop:
  - See How This Fits into Your Application
- A List Called `events` That Contains Objects:
  - Object Appended to List for Each Event
  - Addition is Automatic in This Example
  - List Can Be Checked Continually

An Example Event Loop

```
# Assume that this list gets updated automatically
events = []









# The Event Loop
while True:
    # If the list is empty, skip to next iteration of loop
    if events == []:
        continue

    # If execution reaches here, there is an event to deal with
    event = events[0]
```

An Example Event Loop

```
# Assume that this list gets updated automatically
events = []









# The Event Loop
while True:
    # If the list is empty, skip to next iteration of loop
    if events == []:
        continue

    # If execution reaches here, there is an event to deal with
    event = events[0]

    # If event is a keypress event object
    if event.type == "keypress":
        # Call the keypress event handler
        handle_keypress(event)
```

An Example Event Loop

```
# Assume that this list gets updated automatically
events = []

# Create an event handler
def handle_keypress(event):
    """Print the character associated to the key pressed"""
    print(event.char)

# The Event Loop
while True:
    # If the list is empty, skip to next iteration of loop
    if events == []:
        continue

    # If execution reaches here, there is an event to deal with
    event = events[0]

    # If event is a keypress event object
    if event.type == "keypress":
        # Call the keypress event handler
        handle_keypress(event)
```

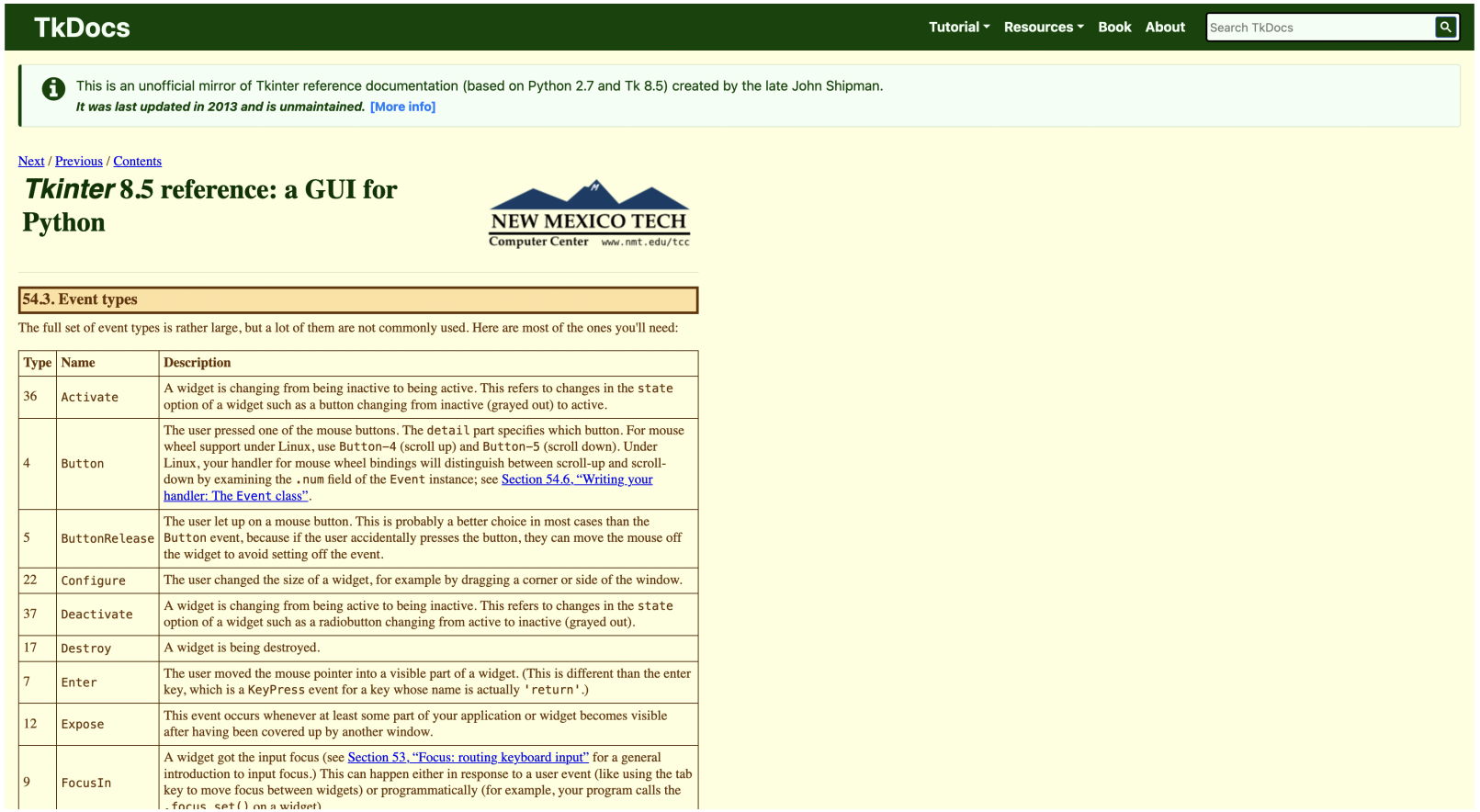
Using `.bind()`

- Calls an Event Handler When an Event Occurs
- Bound Because Handler Called at Each Event
- Takes at Least Two Arguments:
  - Event - `<event_name>`
  - Event Handler - Function Name
- Event Object Passed to Event Handler
- Binding Can be to Any Widget in Application

Mouse Click Events

- `"<Button-1>"` - Primary Mouse Button
- `"<Button-2>"` - Middle Mouse Button
- `"<Button-3>"` - Secondary Mouse Button

TkDocs: Event Type Reference



TkDocs

Tutorial Resources Book About Search TkDocs

This is an unofficial mirror of Tkinter reference documentation (based on Python 2.7 and Tk 8.5) created by the late John Shipman.
It was last updated in 2013 and is unmaintained. [More info]

Next / Previous / Contents

***Tkinter* 8.5 reference: a GUI for Python**

NEW MEXICO TECH
Computer Center www.nmt.edu/tcc

54.3. Event types

The full set of event types is rather large, but a lot of them are not commonly used. Here are most of the ones you'll need:

Type	Name	Description
36	`Activate`	A widget is changing from being inactive to being active. This refers to changes in the `state` option of a widget such as a button changing from inactive (grayed out) to active.
4	`Button`	The user pressed one of the mouse buttons. The `detail` part specifies which button. For mouse wheel support under Linux, use `Button-4` (scroll up) and `Button-5` (scroll down). Under Linux, your handler for mouse wheel bindings will distinguish between scroll-up and scroll-down by examining the `.num` field of the `Event` instance; see Section 54.6, "Writing your handler: The `Event` class".
5	`ButtonRelease`	The user let up on a mouse button. This is probably a better choice in most cases than the `Button` event, because if the user accidentally presses the button, they can move the mouse off the widget to avoid setting off the event.
22	`Configure`	The user changed the size of a widget, for example by dragging a corner or side of the window.
37	`Deactivate`	A widget is changing from being active to being inactive. This refers to changes in the `state` option of a widget such as a radiobutton changing from active to inactive (grayed out).
17	`Destroy`	A widget is being destroyed.
7	`Enter`	The user moved the mouse pointer into a visible part of a widget. (This is different than the enter key, which is a `KeyPress` event for a key whose name is actually `'return'`.)
12	`Expose`	This event occurs whenever at least some part of your application or widget becomes visible after having been covered up by another window.
9	`FocusIn`	A widget got the input focus (see Section 53, "Focus: routing keyboard input" for a general introduction to input focus.) This can happen either in response to a user event (like using the tab key to move focus between widgets) or programmatically (for example, your program calls the `.focus_set()` on a widget).

https://tkdocs.com/shipman/event-types.html

Next: Using `command`

Building a Python GUI Application With `tkinter`

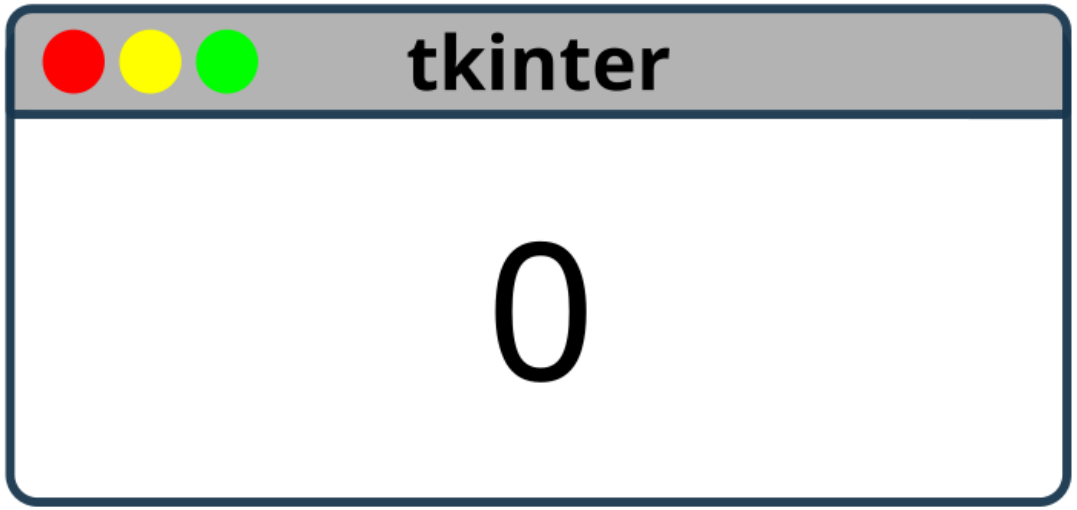
Using `command`

- Provided by `Button` Widgets
- Assigns a Function
- Function Called on Each Button Press

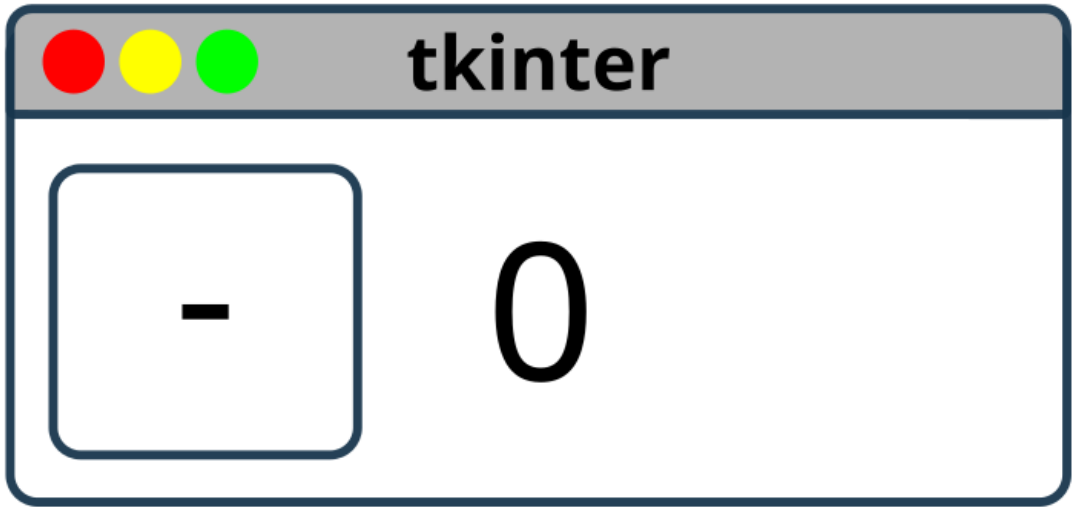
Using `command`: An Example

Using `command`: An Example

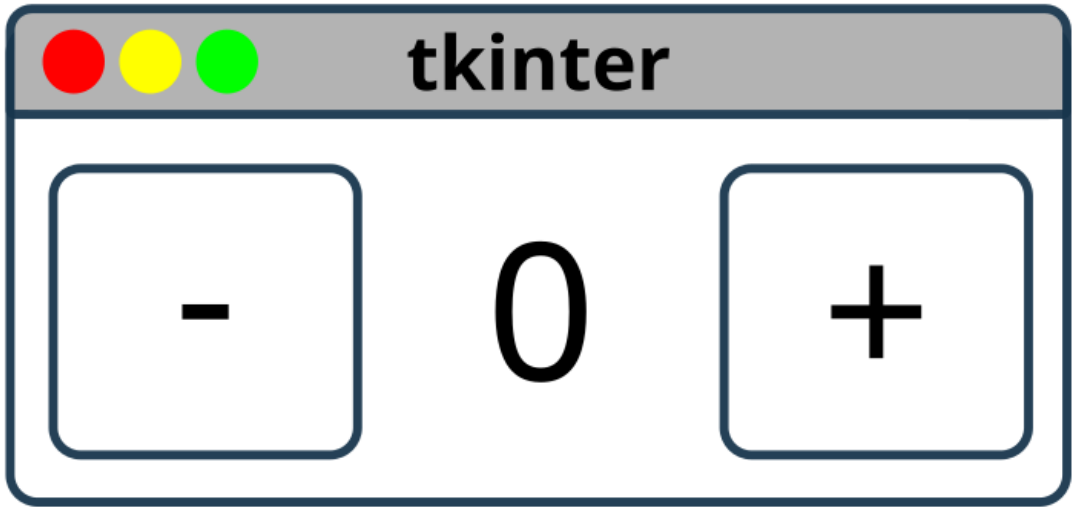
Using `command`: An Example



Using `command`: An Example



Using `command`: An Example



Using `command`: An Example

- Left Button:
  - Decrease The Value by `1`
- Needed Techniques:
  - Getting the Text in `Label`
  - Updating the Text in `Label`

Querying and Updating `Label` Widgets

```
label = tk.Label(text="Hello")
```

- Accessing Text:
  - `text = label["text"]`
- Setting Text:
  - `label["text"] = "Goodbye"`

Transferrable Skills for Future Apps

- Widgets
- Geometry Managers
- Event Handlers

Next: Building A Temperature Converter

Building a Python GUI Application With `tkinter`

1. Getting `tkinter` Up and Running
2. Building Your First GUI Application With `tkinter`
3. Working With Widgets
4. Controlling Layout With Geometry Managers
5. Making Your Applications Interactive
6. **Building A Temperature Converter**
7. Building A Text Editor

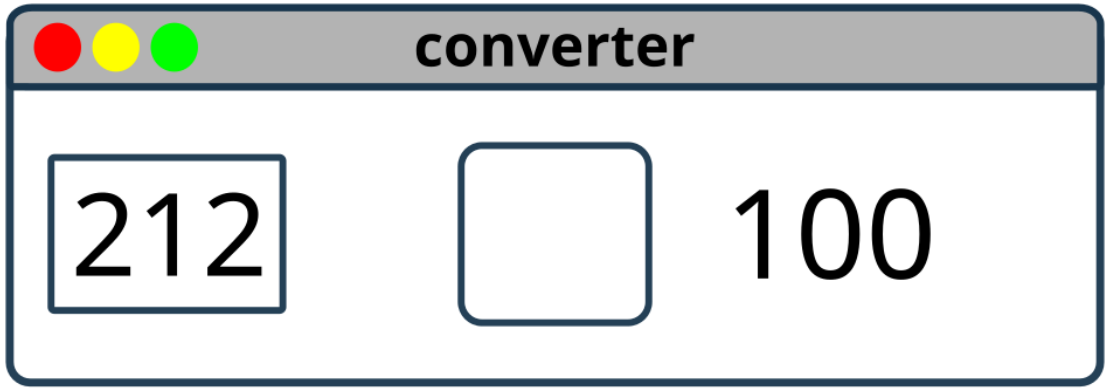
Building A Temperature Converter

- Input in Degrees Fahrenheit
- Convert to Degrees Celsius
- Coded Step-by-Step
- Final Code in Course Materials

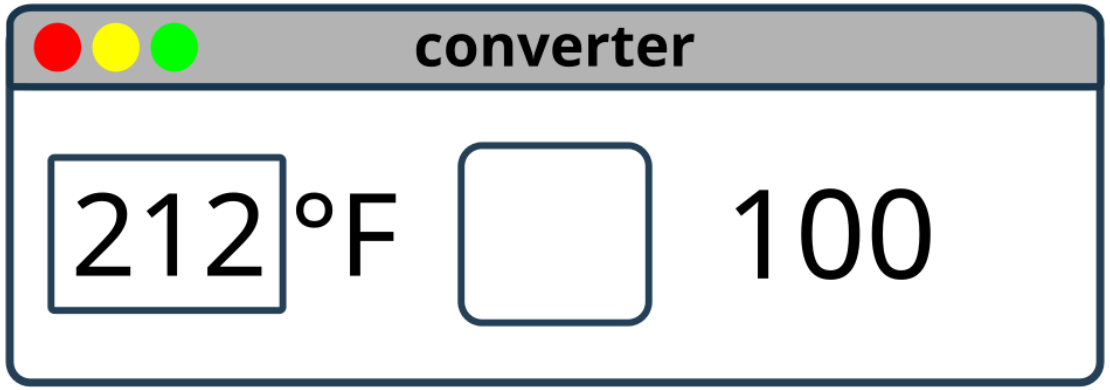
Application Design

- `Entry`:
  - `ent_temperature`
  - Entering the Fahrenheit Value
- `Label`:
  - `lbl_result`
  - Display the Celsius Result
- `Button`:
  - `btn_convert`
  - Reads the Value From the Entry Widget
  - Converts It From Fahrenheit to Celsius
  - Sets the Text of the Label Widget to the Result When Clicked

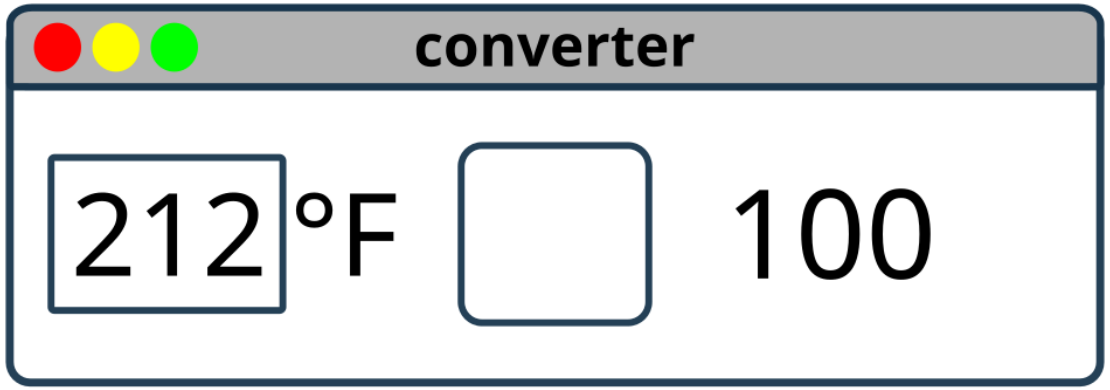
Application Design



Application Design

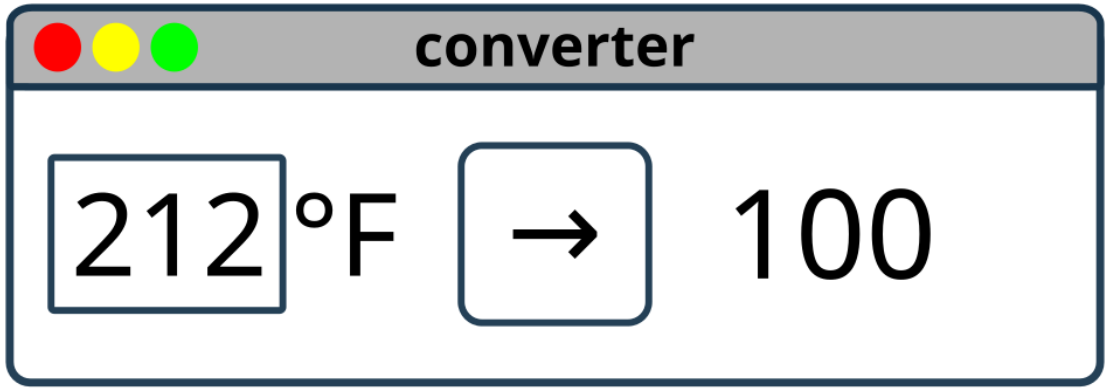


Application Design



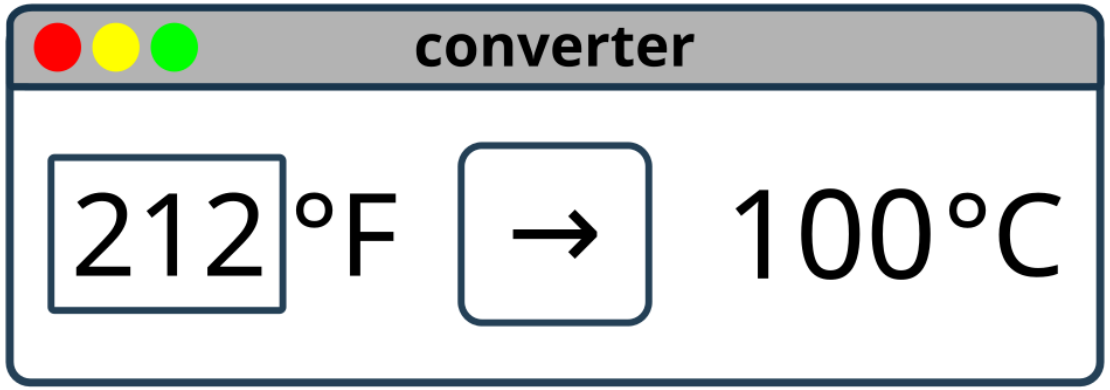

```
lbl_temp = tk.Label(text="\N{DEGREE FAHRENHEIT}")
```

Application Design




```
tk.Button(text="\N{RIGHTWARDS BLACK ARROW}")
```

Application Design




```
lbl_temp = tk.Label(text="\N{DEGREE CELSIUS}")
```

Next: Building A Text Editor

Building a Python GUI Application With `tkinter`

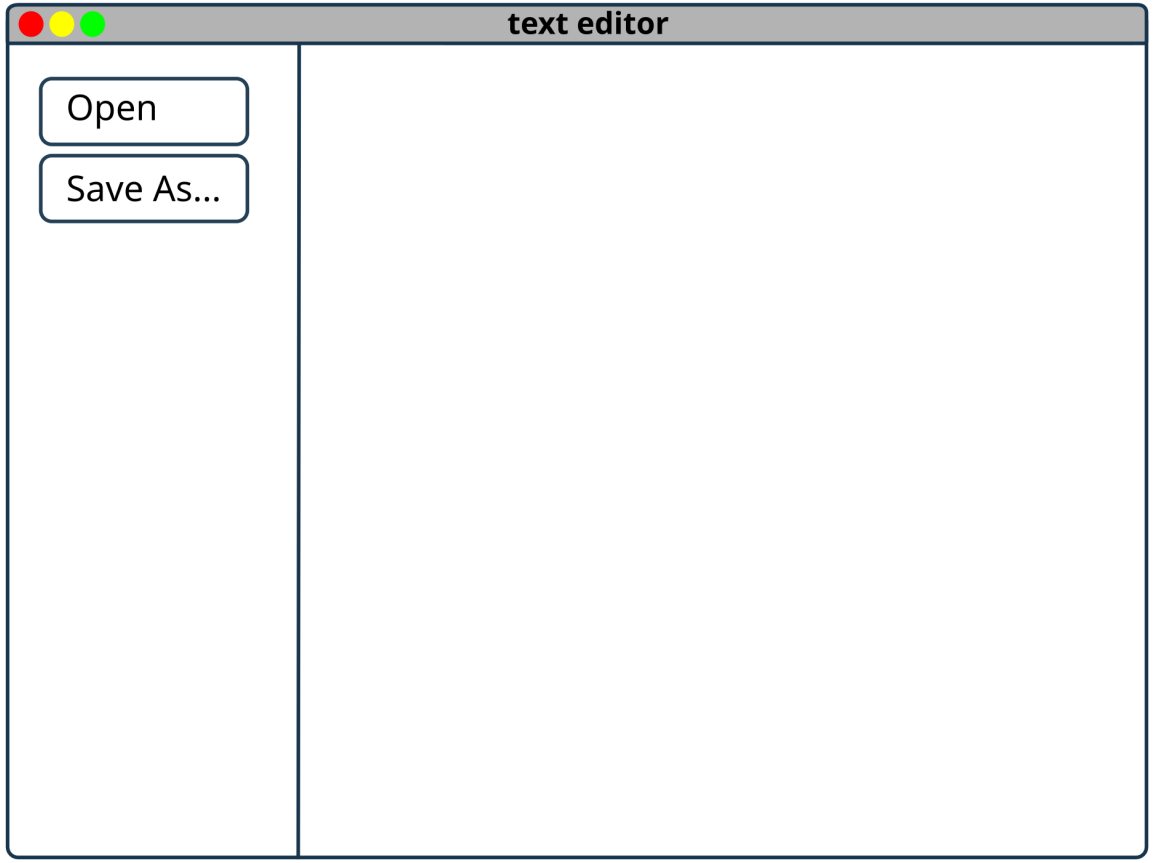
Building A Text Editor

- Create
- Open
- Edit
- Save

Building A Text Editor

- `btn_open`:
  - Opening a File for Editing
- `btn_save`:
  - Saving a File
- `txt_edit`:
  - Creating and Editing the Text File

Building A Text Editor: Application Appearance



Building A Text Editor: Window Specification

- Window Minimum Height: 800 pixels
- `txt_edit` Minimum Width: 800 pixels
- Responsive Layout:
  - `txt_edit` Resizes with Window
  - Button Frame Width Constant

Building A Text Editor: Geometry Manager

- Use `.grid()` Geometry Manager
- One Row
- Two Columns:
  - A Narrow Column on the Left:
    - `btn_open`
    - `btn_save`
  - A Wider Column on the Right:
    - `txt_edit`

Building A Text Editor: Setting Sizes

```
window.rowconfigure(0)
window.columnconfigure(1)
```

Building A Text Editor: Setting Sizes

```
window.rowconfigure(0, minsize=800)
window.columnconfigure(1, minsize=800)
```

Building A Text Editor: Setting Sizes

```
window.rowconfigure(0, minsize=800, weight=1)
window.columnconfigure(1, minsize=800, weight=1)
```

Building A Text Editor: Button Layout

- Frame Widget - `frm_buttons`
- Buttons Stacked Vertically:
  - `btn_open` on Top
- Use `.grid()` Geometry Manager

Next: Adding Load Functionality

Building a Python GUI Application With `tkinter`

Adding Load Functionality

- Layout Of Editor Complete - 
- Load Functionality:
  - `btn_open`:
    - Show File Open Dialog
    - Allow User to Select a File
    - Open File
    - Set Contents of `txt_edit`

Next: Adding Save Functionality

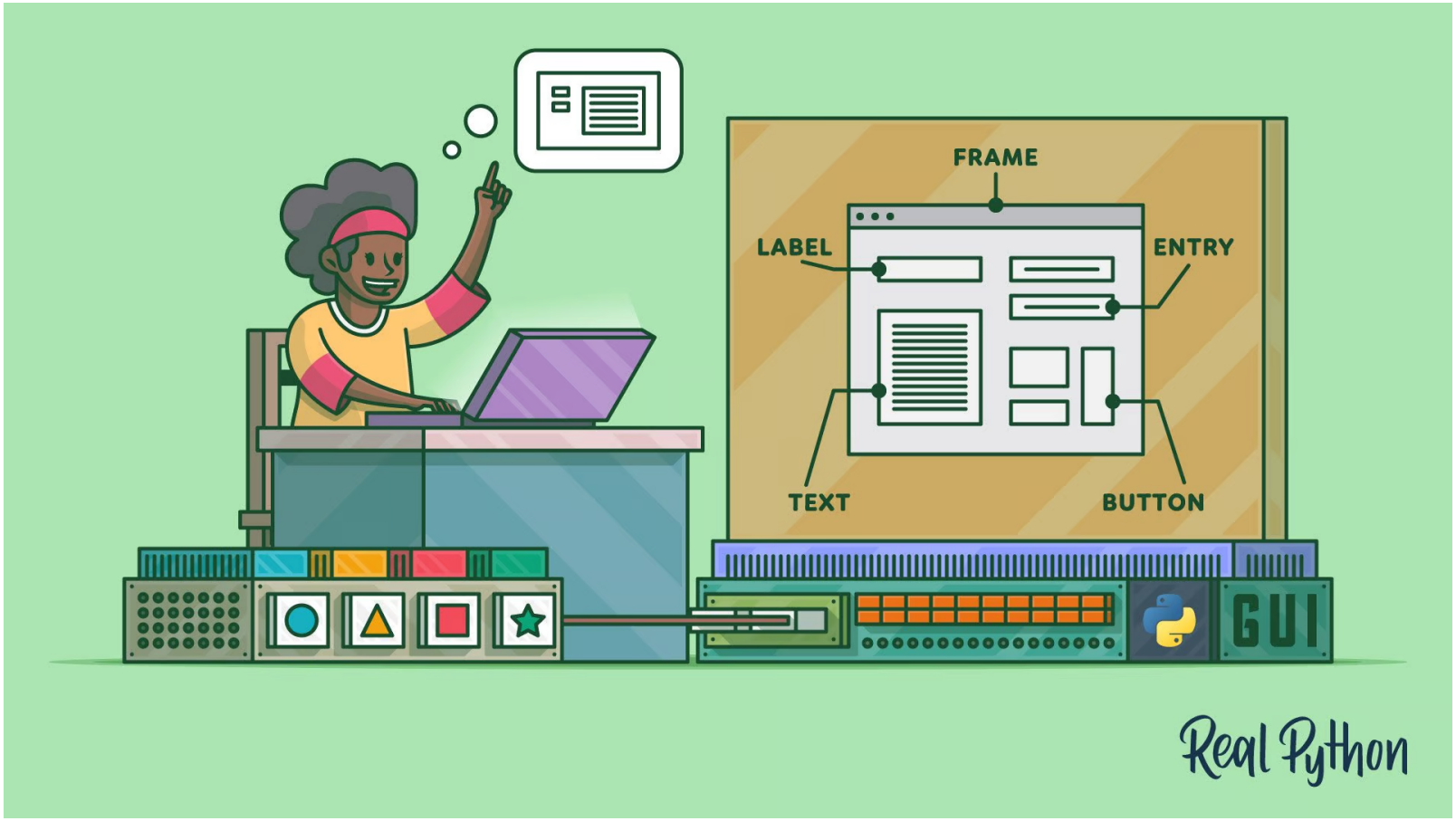
Building a Python GUI Application With `tkinter`

Adding Save Functionality

- `btn_open` Working - ✅
- Save Functionality:
  - `btn_save`:
    - Show Save File Dialog
    - Allow User to Select Save Location and Filename
    - Use `asksaveasfilename` from `tkinter.filedialog`
    - Extract Contents of `txt_edit`
    - Write to Selected File

Next: Summary

Building a Python GUI Application With `tkinter` : Summary



Summary

- Built Two GUI Applications
- Applied Learned Skills
- Ready to:
  - Add to Application Functionality
  - Tackle New Applications
- `tkinter`:
  - Built-In to Python
  - Relatively Easy to Create Applications

Summary

- Use Five `tkinter` Widgets:
  - `Label`
  - `Button`
  - `Entry`
  - `Text`
  - `Frame`
- Control Your Application Layout With Geometry Managers
- Make Your Applications Interactive
- Build Small Applications

Summary