Hey there!

Welcome to the **PyTermGUI** documentation!

For a general overview of the framework, follow the navigation at the bottom. For a set of working examples, check out the walkthroughs section. For a complete look at the API, see our reference!

index image

So what is a Terminal User Interface (or TUI) anyways?

A Terminal User Interface is, in general terms, a GUI in your terminal. Due to the terminal's attributes & limitations it can *only* use plain text to display content, and has to arrange itself on the terminal's character grid.

In effect, this means terminal applications are:

- More constrained, which can often lead to more interesting solutions around known problems
- More standardized (though not enough)
- *Much* faster to execute
- *Way cooler lookin'*

If all that is true, why are TUIs so rare?

Historically, you had 2 options when writing a TUI:

- Create your own framework with its own rendering pipeline
- Use curses or ncurses

Neither of these are simple to do; drawing in the terminal requires knowledge of a bunch of scattered standards and precedents, which, for the most part, are only available through *archaic* documentation.

Things *have* gotten a lot better in recent years. Projects like Jexer and notcurses have shown how capable terminals are, and projects like bubbletea and tui-rs have made simplified and modernized TUI creation.

How do we help the situation?

PTG provides abstractions for the low-level interactions with the terminal, and a modular widget system that is built ontop.

Our goals are the following:

- Versatility
- Expressive, readable but compact code
- No pointless abstractions
- Minimal dependencies

Because of the way the framework is built, you can implement the input-key-mouse-draw loop used internally by WindowManager in {insert linecount} lines!

Widget layouts are explicit from the syntax of their creation. Widgets are rendered line-by-line, using the plain strings they return. There is built-in support for keyboard & mouse inputs on all widgets, as well as the underlying low-level API that is used to make it all work.

This means that you, the user, can create TUI applications with workflows that rival the web in simplicity, without having to deal with the *massive* fragmentation, framework war, and *shudders* CSS.