Overview

This lesson explains how commands are run in Tkinter, how widgets trigger your code, and how callbacks help keep the event loop responsive and safe.

Important Information

What a Command Is

In Tkinter, a **command** is a function that is run in response to a user action. Common examples include clicking a button, toggling a checkbutton, or selecting a menu item.

Commands do not run on their own. They are triggered by the event loop when the user interacts with a widget.

Linking a Command to a Widget

Many interactive widgets accept a `command` option. This option is given a function **reference**, not a function call.

```
from tkinter import *
from tkinter import ttk

root = Tk()

def submit():
    print("Submitted")

button = ttk.Button(root, text="Submit", command=submit)
button.grid(row=0, column=0)

root.mainloop()
```

The key rule is that you **do not use parentheses** when assigning the command. Writing `command=submit()` would call the function immediately instead of waiting for the user action.

What Happens When a Command Runs

When the user activates a widget, Tkinter places an event into the event queue. The event loop eventually processes that event and calls the linked command function.

The function briefly takes control, runs its code, and then returns control back to the event loop. This cycle is what keeps the program responsive.

Using Callbacks to Protect the Event Loop

Command functions run inside the event loop. If a command takes too long, the event loop becomes blocked and the interface freezes.

For this reason, callbacks should:

- Do a small amount of work
- Return control to the event loop quickly
- Avoid long loops, `sleep`, or blocking operations

A problematic callback looks like this:

```
def bad_command():
    for i in range(100_000_000):
        pass
```

A safer pattern is to break work into steps and schedule each step using `after`.

```
step = 0

def run_step():
    global step
    step += 1
    print("Step", step)

    if step < 10:
        root.after(100, run_step)
```

This allows the event loop to handle screen updates and user input between steps.

Commands vs Event Bindings

Commands are simple and safe. They are best used when:

- A widget has one clear action
- You do not need information about the event itself

Event bindings are more flexible but more complex. They pass an event object and are used when you need details like mouse position or key pressed.

For most buttons and basic interactions, commands are the preferred approach.

Best Practices for Command Functions

Command and callback functions should follow clear rules to avoid bugs and freezes.

Functions used as commands should:

- Take no parameters
- Have a clear, single responsibility
- Be short and easy to read
- Avoid long-running work
- Schedule additional work using `after` if needed

```
def save_file():
    validate_inputs()
    start_save_process()
```

If logic becomes complex, move it into helper functions rather than cramming everything into the command itself.

Keeping Commands Predictable

A command should never assume it is the only thing running. The event loop may process other events immediately before or after it. Writing small, predictable command functions makes your program easier to debug and keeps the interface responsive as it grows.

Understanding how commands interact with the event loop is essential for writing reliable GUI programs in Tkinter.