Overview

A `button`, unlike a frame or label, is very much there to interact with. Users press a button to perform an action. Like labels, they can display text or images, but accept additional options to change their behavior.

Important Information

Creating a Button

Buttons are created using the `ttk.Button` class:

```
# Create a button using the button class
# parent represents whatever container you want to put it in. This can be
the root, or a frame.
button = ttk.Button(parent, text='Okay', command=submitForm)
```

Typically, their **contents** *(the text to diaplay)* and **command callback** *(the function the button will run)* are specified at the same time the button is created. As with other widgets, buttons accept several configuration options to alter their appearance and behavior, including the standard `style` option.

Text or Image

Buttons take the same `text`, `textvariable` *(less commonly used)*, `image`, and `compound` configuration options as labels. These control whether the button displays text and/or an image.

For example:

```
# Create the button
button = ttk.Button(root, command='my_function')
# Set the text of the button
button['text'] = "Click Me!"
# Create an image variable to store the image
image = PhotoImage(file='picture.png')
# Set the button's image to the image variable
button['image'] = image
# Set the compound configuration for how the text and image appear
together
button['compound'] = 'center'
# Display the button
button.grid()
```

In the above example, it includes image and text, but you can choose to only use one or the other instead as well.

Buttons have a `default` configuration option. If specified as `active`, this tells Tk that the button is the default button in the user interface; otherwise, it is `normal`. Default buttons are invoked if users hit the `Return` or `Enter` key. Some platforms and styles will draw this default button with a different border or highlight.

For example:

```
# Setting defualt to "active" makes it so that pressing Enter or Return
counts as pressing the button
button['default'] = 'active'
```

Note that setting this configuration option doesn't create an event binding that will make the "Return" or "Enter" key activate the button; you have to do that yourself.

The Command Callback

The `command` option connects the button's action and your application. When a user presses the button, the script or function provided by the option is evaluated by the interpreter.

For example:

```
# Set the command callback of the button
button['command'] = my_function
```

Button State

Buttons and many other widgets start off in a *normal* state. A button in its normal state will respond to mouse movements, can be pressed, and will invoke its command callback. Buttons can also be put into a *disabled* state, where the button is greyed out, does not respond to mouse movements, and cannot be pressed. Your program would disable the button when its command is not applicable based on the state of the application at a given point in time.

```
# Disable the button
button.state(['disabled'])
# Clear the disabled flag (enable the button again)
button.state(['!disabled'])
```

All themed widgets maintain an internal state, represented as a series of binary flags. Each flag can either be set *(on)* or cleared *(off)*. You can set or clear these different flags, and check the current setting using the state and instate methods. Buttons make use of the disabled flag to control whether or not users can press the button. For example:

```
# Checks if the button is disabled
# Returns True if the button is disabled
# Returns False if the button is not disabled
button.instate(['disabled'])
# Checks if the button is not disabled
# Returns True if the button is not disabled
# Returns False if the button is disabled
button.instate(['!disabled'])
```

The full list of state flags available to themed widgets is: `active`, `disabled`, `focus`, `pressed`, `selected`, `background`, `readonly`, `alternate`, and `invalid`. These are described in the themed widget reference. While all widgets have the same set of state flags, **not all states are meaningful for all widgets**.

Copy, Change, Challenge

Copy

```
from tkinter import *
from tkinter import ttk

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

def on_button_click():
    print("Button was clicked!")

# Create the button
button = ttk.Button(root, text="Click Me", command=on_button_click)

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

root.mainloop()
```

Change

Modify the program so that:

- The button text is changed to **"Submit"**
- The button is set as the **default button**
- The button is **disabled** when the program starts
- After you confirm the program runs, **re-enable** the button by removing the disabled state

Challenge

Create a program with **two buttons**:

- The first button should:
    - Display **text and an image** using the `compound` option
    - Print a message to the terminal when clicked
- The second button should:
    - Start in a **disabled** state
    - Become **enabled** only after the first button has been clicked
- Place both buttons using `grid` in different rows