CustomTkinter DOCUMENTATION

INDEX:

Sure! Here are the attributes you can use with the `CTkButton` widget in CustomTkinter:

1. **text**: The text displayed on the button.
2. **command**: The function to be called when the button is clicked.
3. **width**: The width of the button.
4. **height**: The height of the button.
5. **fg_color**: The foreground color of the button.
6. **bg_color**: The background color of the button.
7. **hover_color**: The color of the button when the mouse hovers over it.
8. **border_color**: The color of the button’s border.
9. **border_width**: The width of the button’s border.
10. **corner_radius**: The radius of the button’s corners.
11. **font**: The font of the button’s text.
12. **text_color**: The color of the button’s text.
13. **state**: The state of the button (e.g., ‘normal’, ‘disabled’).
14. **image**: An image to display on the button.
15. **compound**: Specifies how to display the image relative to the text (e.g., ‘left’, ‘right’, ‘top’, ‘bottom’, ‘center’).

You can configure these attributes using the `configure` method or directly when creating the button. For example:

```
import customtkinter as ctk

root = ctk.CTk()

button = ctk.CTkButton(
    master=root,
    text="Click Me",
    command=lambda: print("Button clicked!"),
    width=100,
    height=50,
    fg_color="blue",
    bg_color="white",
    hover_color="lightblue",
    border_color="black",
    border_width=2,
    corner_radius=10,
    font=("Arial", 14),
    text_color="white",
    state="normal",
    image=None,
    compound="center"
)

button.pack(pady=20)

root.mainloop()
```

Here are the attributes you can use with the `CTkEntry` widget in CustomTkinter:

1. **textvariable**: Associates a Tkinter variable (e.g., `StringVar`) with the entry widget.
2. **width**: The width of the entry widget.
3. **height**: The height of the entry widget.
4. **fg_color**: The foreground color of the entry widget.

5. **bg_color**: The background color of the entry widget.
6. **border_color**: The color of the entry widget's border.
7. **border_width**: The width of the entry widget's border.
8. **corner_radius**: The radius of the entry widget's corners.
9. **font**: The font used for the text in the entry widget.
10. **text_color**: The color of the text in the entry widget.
11. **placeholder_text**: The placeholder text displayed when the entry is empty.
12. **show**: A character to display instead of the actual characters (useful for password fields).
13. **state**: The state of the entry widget (e.g., 'normal', 'disabled').

You can configure these attributes using the `configure` method or directly when creating the entry widget. Here's an example:

```
import customtkinter as ctk

root = ctk.CTk()

entry = ctk.CTkEntry(
    master=root,
    textvariable=ctk.StringVar(),
    width=200,
    height=30,
    fg_color="white",
    bg_color="gray",
    border_color="black",
    border_width=2,
    corner_radius=10,
    font=("Arial", 14),
    text_color="black",
    placeholder_text="Enter text here...",
    show="*",
    state="normal"
)

entry.pack(pady=20)

root.mainloop()
```

Here are the attributes you can use with the `CTkCheckBox` widget in CustomTkinter:

1. **text**: The text displayed next to the checkbox.
2. **command**: The function to be called when the checkbox is toggled.
3. **variable**: Associates a Tkinter variable (e.g., `IntVar`) with the checkbox.
4. **onvalue**: The value assigned to the variable when the checkbox is checked.
5. **offvalue**: The value assigned to the variable when the checkbox is unchecked.
6. **width**: The width of the checkbox.
7. **height**: The height of the checkbox.
8. **fg_color**: The foreground color of the checkbox.
9. **bg_color**: The background color of the checkbox.
10. **hover_color**: The color of the checkbox when the mouse hovers over it.
11. **border_color**: The color of the checkbox's border.
12. **border_width**: The width of the checkbox's border.
13. **corner_radius**: The radius of the checkbox's corners.
14. **font**: The font of the checkbox's text.
15. **text_color**: The color of the checkbox's text.
16. **state**: The state of the checkbox (e.g., 'normal', 'disabled').

You can configure these attributes using the `configure` method or directly when creating the checkbox. Here’s an example:

```
import customtkinter as ctk

root = ctk.CTk()

checkbox_var = ctk.IntVar()

checkbox = ctk.CTkCheckBox(
    master=root,
    text="I agree",
    command=lambda: print("Checkbox toggled!"),
    variable=checkbox_var,
    onvalue=1,
    offvalue=0,
    width=200,
    height=30,
    fg_color="blue",
    bg_color="white",
    hover_color="lightblue",
    border_color="black",
    border_width=2,
    corner_radius=10,
    font=("Arial", 14),
    text_color="black",
    state="normal"
)

checkbox.pack(pady=20)

root.mainloop()
```

Here are the attributes you can use with the `CTkComboBox` widget in CustomTkinter:

1. **values**: A list of values to display in the combobox.
2. **textvariable**: Associates a Tkinter variable (e.g., `StringVar`) with the combobox.
3. **width**: The width of the combobox.
4. **height**: The height of the combobox.
5. **fg_color**: The foreground color of the combobox.
6. **bg_color**: The background color of the combobox.
7. **border_color**: The color of the combobox’s border.
8. **border_width**: The width of the combobox’s border.
9. **corner_radius**: The radius of the combobox’s corners.
10. **font**: The font used for the text in the combobox.
11. **text_color**: The color of the text in the combobox.
12. **state**: The state of the combobox (e.g., ‘normal’, ‘readonly’, ‘disabled’).
13. **justify**: The alignment of the text within the combobox (e.g., ‘left’, ‘center’, ‘right’).
14. **postcommand**: A function to be called right before the dropdown list is shown.

You can configure these attributes using the `configure` method or directly when creating the combobox. Here’s an example:

```
import customtkinter as ctk

root = ctk.CTk()

combobox_var = ctk.StringVar()
```

```
combobox = ctk.CTkComboBox(
    master=root,
    values=["Option 1", "Option 2", "Option 3"],
    textvariable=combobox_var,
    width=200,
    height=30,
    fg_color="white",
    bg_color="gray",
    border_color="black",
    border_width=2,
    corner_radius=10,
    font=("Arial", 14),
    text_color="black",
    state="normal",
    justify="center",
    postcommand=lambda: print("Dropdown opened!")
)

combobox.pack(pady=20)

root.mainloop()
```

Here are the attributes you can use with the `CTkProgressBar` widget in CustomTkinter:

1. **value**: The current value of the progress bar.
2. **maximum**: The maximum value of the progress bar.
3. **mode**: The mode of the progress bar (e.g., ‘determinate’, ‘indeterminate’).
4. **width**: The width of the progress bar.
5. **height**: The height of the progress bar.
6. **fg_color**: The foreground color of the progress bar.
7. **bg_color**: The background color of the progress bar.
8. **border_color**: The color of the progress bar’s border.
9. **border_width**: The width of the progress bar’s border.
10. **corner_radius**: The radius of the progress bar’s corners.
11. **orientation**: The orientation of the progress bar (e.g., ‘horizontal’, ‘vertical’).

You can configure these attributes using the `configure` method or directly when creating the progress bar. Here’s an example:

```
import customtkinter as ctk

root = ctk.CTk()

progress_bar = ctk.CTkProgressBar(
    master=root,
    value=50,
    maximum=100,
    mode="determinate",
    width=200,
    height=20,
    fg_color="blue",
    bg_color="white",
    border_color="black",
    border_width=2,
    corner_radius=10,
    orientation="horizontal"
)

progress_bar.pack(pady=20)
```

```
root.mainloop()
```

Here are the attributes you can use with the `CTkRadioButton` widget in CustomTkinter:

1. **text**: The text displayed next to the radio button.
2. **command**: The function to be called when the radio button is selected.
3. **variable**: Associates a Tkinter variable (e.g., `IntVar` or `StringVar`) with the radio button.
4. **value**: The value assigned to the variable when the radio button is selected.
5. **width**: The width of the radio button.
6. **height**: The height of the radio button.
7. **fg_color**: The foreground color of the radio button.
8. **bg_color**: The background color of the radio button.
9. **hover_color**: The color of the radio button when the mouse hovers over it.
10. **border_color**: The color of the radio button's border.
11. **border_width**: The width of the radio button's border.
12. **corner_radius**: The radius of the radio button's corners.
13. **font**: The font of the radio button's text.
14. **text_color**: The color of the radio button's text.
15. **state**: The state of the radio button (e.g., 'normal', 'disabled').

You can configure these attributes using the `configure` method or directly when creating the radio button. Here's an example:

```
import customtkinter as ctk

root = ctk.CTk()

radio_var = ctk.IntVar()

radio_button1 = ctk.CTkRadioButton(
    master=root,
    text="Option 1",
    command=lambda: print("Option 1 selected!"),
    variable=radio_var,
    value=1,
    width=200,
    height=30,
    fg_color="blue",
    bg_color="white",
    hover_color="lightblue",
    border_color="black",
    border_width=2,
    corner_radius=10,
    font=("Arial", 14),
    text_color="black",
    state="normal"
)

radio_button2 = ctk.CTkRadioButton(
    master=root,
    text="Option 2",
    command=lambda: print("Option 2 selected!"),
    variable=radio_var,
    value=2,
    width=200,
    height=30,
    fg_color="blue",
    bg_color="white",
    hover_color="lightblue",
    border_color="black",
    border_width=2,
    corner_radius=10,
    font=("Arial", 14),
    text_color="black",
    state="normal"
)

radio_button1.pack(pady=10)
radio_button2.pack(pady=10)

root.mainloop()
```

Creating scrollable frames in CustomTkinter involves using a `Canvas` widget along with a `Scrollbar`. Here's how you can set up a scrollable frame:

1. **Create a Canvas**: This will act as the container for the scrollable content.
2. **Add a Scrollbar**: Attach a scrollbar to the canvas.
3. **Create an Inner Frame**: Place a frame inside the canvas to hold the actual content.
4. **Configure Scrolling**: Ensure the canvas scrolls when the inner frame's size changes.

Here's an example:

```
import customtkinter as ctk
from tkinter import Canvas, Scrollbar, Frame, VERTICAL, HORIZONTAL, NW

class ScrollableFrame(ctk.CTkFrame):
    def __init__(self, master, **kwargs):
        super().__init__(master, **kwargs)

        # Create a canvas
        self.canvas = Canvas(self)
        self.canvas.grid(row=0, column=0, sticky="nsew")

        # Add vertical scrollbar
        self.v_scrollbar = Scrollbar(self, orient=VERTICAL, command=self.canvas.yview)
        self.v_scrollbar.grid(row=0, column=1, sticky="ns")

        # Add horizontal scrollbar
        self.h_scrollbar = Scrollbar(self, orient=HORIZONTAL,
command=self.canvas.xview)
        self.h_scrollbar.grid(row=1, column=0, sticky="ew")

        # Configure canvas to work with scrollbars
        self.canvas.configure(yscrollcommand=self.v_scrollbar.set,
xscrollcommand=self.h_scrollbar.set)

        # Create an inner frame
        self.inner_frame = Frame(self.canvas)
        self.canvas.create_window((0, 0), window=self.inner_frame, anchor=NW)

        # Ensure scrolling works
        self.inner_frame.bind("<Configure>", lambda event:
self.canvas.configure(scrollregion=self.canvas.bbox("all")))

        # Make the frame expandable
        self.grid_rowconfigure(0, weight=1)
        self.grid_columnconfigure(0, weight=1)

# Example usage
root = ctk.CTk()
```

```
scrollable_frame = ScrollableFrame(root, width=400, height=300)
scrollable_frame.pack(fill="both", expand=True)

# Add some sample content
for i in range(50):
    label = ctk.CTkLabel(scrollable_frame.inner_frame, text=f"Label {i}")
    label.pack()

root.mainloop()
```

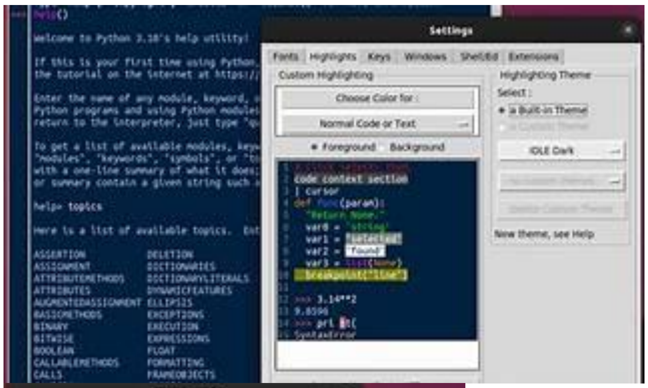
Here are the attributes you can use with the `CTkSegmentedButton` widget in CustomTkinter:

1. **values**: A list of string values for the buttons.
2. **command**: The function to be called when a button is clicked.
3. **variable**: Associates a Tkinter variable (e.g., `StringVar`) with the segmented button.
4. **width**: The width of the segmented button.
5. **height**: The height of the segmented button.
6. **corner_radius**: The radius of the segmented button's corners.
7. **border_width**: The width of the segmented button's border.
8. **fg_color**: The foreground color around the buttons.
9. **selected_color**: The color of the selected button.
10. **selected_hover_color**: The hover color of the selected button.
11. **unselected_color**: The color of the unselected buttons.
12. **unselected_hover_color**: The hover color of the unselected buttons.
13. **text_color**: The color of the text on the buttons.
14. **text_color_disabled**: The color of the text when the button is disabled.
15. **font**: The font of the button text.
16. **state**: The state of the segmented button (e.g., 'normal', 'disabled').
17. **dynamic_resizing**: Enable/disable automatic resizing when text is too big to fit.

You can configure these attributes using the `configure` method or directly when creating the segmented button. Here's an example:

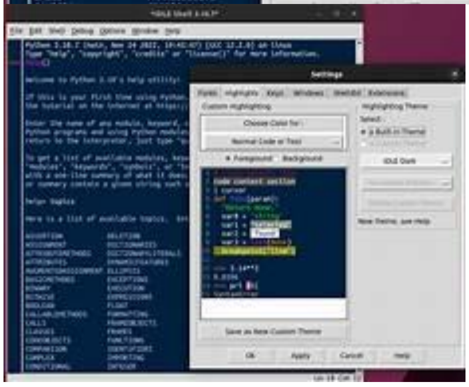
```
import customtkinter as ctk

def segmented_button_callback(value):
    print("Segmented button clicked:", value)

root = ctk.CTk()

segmented_button_var = ctk.StringVar(value="Value 1")

segmented_button = ctk.CTkSegmentedButton(
    master=root,
    values=["Value 1", "Value 2", "Value 3"],
    command=segmented_button_callback,
    variable=segmented_button_var,
    width=300,
    height=50,
    corner_radius=10,
    border_width=2,
    fg_color=("lightgray", "darkgray"),
    selected_color=("blue", "darkblue"),
    selected_hover_color=("lightblue", "darkblue"),
    unselected_color=("white", "gray"),
    unselected_hover_color=("lightgray", "darkgray"),
    text_color=("black", "white"),
    text_color_disabled=("gray", "darkgray"),
    font=("Arial", 14),
    state="normal",
    dynamic_resizing=True
)

segmented_button.pack(pady=20)

root.mainloop()
```

PG 9



```
>>> help()

Welcome to Python 3.10's help utility!

If this is your first time using Python,
the tutorial on the Internet at https://

Enter the name of any module, keyword, o
Python programs and using Python modules
return to the interpreter, just type "qu

To get a list of available modules, keyw
"modules", "keywords", "symbols", or "to
with a one-line summary of what it does;
or summary contain a given string such a

help> topics

Here is a list of available topics.  Ent

ASSERTION           DELETION
ASSIGNMENT          DICTIONARIES
ATTRIBUTEMETHODS    DICTIONARYLITERALS
ATTRIBUTES          DYNAMICFEATURES
AUGMENTEDASSIGNMENT ELLIPSIS
BASICMETHODS        EXCEPTIONS
BINARY              EXECUTION
BITWISE             EXPRESSIONS
BOOLEAN             FLOAT
CALLABLEMETHODS     FORMATTING
CALLS               FRAMEOBJECTS
```

```
code context section
| cursor
def func(param):
    "Return None."
    var0 = 'string'
    var1 = 'selected'
    var2 = 'found'
    var3 = list(None)
breakpoint("line")

>>> 3.14**2
9.8596
>>> prin t(
SyntaxError
```



```
Python 3.10.7 (main, Nov 24 2022, 19:45:47) [GCC 12.2.0] on linux
Type "help", "copyright", "credits" or "license()" for more information.
>>> help()

Welcome to Python 3.10's help utility!
```

Explore

Here are the attributes you can use with the `CTkSlider` widget in CustomTkinter:

1. **from_**: The starting value of the slider.
2. **to**: The ending value of the slider.
3. **variable**: Associates a Tkinter variable (e.g., `DoubleVar`) with the slider.
4. **command**: The function to be called when the slider's value changes.
5. **width**: The width of the slider.
6. **height**: The height of the slider.
7. **fg_color**: The foreground color of the slider.
8. **bg_color**: The background color of the slider.
9. **progress_color**: The color of the progress bar.
10. **border_color**: The color of the slider's border.
11. **border_width**: The width of the slider's border.
12. **corner_radius**: The radius of the slider's corners.
13. **orientation**: The orientation of the slider (e.g., 'horizontal', 'vertical').
14. **state**: The state of the slider (e.g., 'normal', 'disabled').

You can configure these attributes using the `configure` method or directly when creating the slider. Here's an example:

```
import customtkinter as ctk

def slider_callback(value):
    print("Slider value:", value)
```

```
root = ctk.CTk()

slider = ctk.CTkSlider(
    master=root,
    from_=0,
    to=100,
    variable=ctk.DoubleVar(),
    command=slider_callback,
    width=300,
    height=20,
    fg_color="gray",
    bg_color="white",
    progress_color="blue",
    border_color="black",
    border_width=2,
    corner_radius=10,
    orientation="horizontal",
    state="normal"
)

slider.pack(pady=20)

root.mainloop()
```

Here are the attributes you can use with the `CTkSwitch` widget in CustomTkinter:

1. **text**: The text displayed next to the switch.
2. **command**: The function to be called when the switch is toggled.
3. **variable**: Associates a Tkinter variable (e.g., `StringVar`) with the switch.
4. **onvalue**: The value assigned to the variable when the switch is on.
5. **offvalue**: The value assigned to the variable when the switch is off.
6. **width**: The width of the switch.
7. **height**: The height of the switch.
8. **switch_width**: The width of the switch button.
9. **switch_height**: The height of the switch button.
10. **corner_radius**: The radius of the switch's corners.
11. **border_width**: The width of the switch's border.
12. **fg_color**: The foreground color of the switch.
13. **bg_color**: The background color of the switch.
14. **border_color**: The color of the switch's border.
15. **progress_color**: The color of the switch when it is on.
16. **button_color**: The color of the switch button.
17. **button_hover_color**: The hover color of the switch button.
18. **hover_color**: The hover color of the switch.
19. **text_color**: The color of the text next to the switch.
20. **font**: The font of the text next to the switch.
21. **state**: The state of the switch (e.g., 'normal', 'disabled').

You can configure these attributes using the `configure` method or directly when creating the switch. Here's an example:

```
import customtkinter as ctk

def switch_event():
    print("Switch toggled, current value:", switch_var.get())
```

```
root = ctk.CTk()

switch_var = ctk.StringVar(value="on")

switch = ctk.CTkSwitch(
    master=root,
    text="CTkSwitch",
    command=switch_event,
    variable=switch_var,
    onvalue="on",
    offvalue="off",
    width=100,
    height=40,
    switch_width=30,
    switch_height=20,
    corner_radius=10,
    border_width=2,
    fg_color="lightgray",
    bg_color="white",
    border_color="black",
    progress_color="blue",
    button_color="white",
    button_hover_color="lightblue",
    hover_color="lightgray",
    text_color="black",
    font=("Arial", 14),
    state="normal"
)

switch.pack(pady=20)

root.mainloop()
```

Here are the attributes you can use with the `CTkTabview` widget in CustomTkinter:

1. **master**: The parent widget (e.g., root, frame, top-level).
2. **width**: The width of the tab view.
3. **height**: The height of the tab view.
4. **corner_radius**: The radius of the tab view’s corners.
5. **border_width**: The width of the tab view’s border.
6. **fg_color**: The foreground color of the tab view and the tabs.
7. **border_color**: The color of the tab view’s border.
8. **segmented_button_fg_color**: The foreground color of the segmented button.
9. **segmented_button_selected_color**: The color of the selected segmented button.
10. **segmented_button_selected_hover_color**: The hover color of the selected segmented button.
11. **segmented_button_unselected_color**: The color of the unselected segmented buttons.
12. **segmented_button_unselected_hover_color**: The hover color of the unselected segmented buttons.
13. **text_color**: The color of the text on the segmented buttons.
14. **text_color_disabled**: The color of the text when the segmented button is disabled.
15. **command**: The function to be called when a segmented button is clicked.
16. **anchor**: The position of the segmented button (e.g., ‘n’, ‘nw’, ‘ne’, ‘sw’, ‘s’, ‘se’).
17. **state**: The state of the tab view (e.g., ‘normal’, ‘disabled’).

You can configure these attributes using the `configure` method or directly when creating the tab view. Here’s an example:

```
import customtkinter as ctk
```

```
class MyTabView(ctk.CTkTabview):
    def __init__(self, master, **kwargs):
        super().__init__(master, **kwargs)
        # Create tabs
        self.add("Tab 1")
        self.add("Tab 2")
        # Add widgets on tabs
        self.label = ctk.CTkLabel(master=self.tab("Tab 1"), text="Content of Tab 1")
        self.label.grid(row=0, column=0, padx=20, pady=10)

class App(ctk.CTk):
    def __init__(self):
        super().__init__()
        self.tab_view = MyTabView(master=self, width=400, height=300, corner_radius=10, 
border_width=2)
        self.tab_view.grid(row=0, column=0, padx=20, pady=20)

app = App()
app.mainloop()
```

This example creates a tab view with two tabs and adds a label to the first tab. You can add any widgets to the tabs just like you would with a `CTkFrame`[12].

Here are the attributes you can use with the `CTkTextbox` widget in CustomTkinter:

1. **width**: The width of the text box.
2. **height**: The height of the text box.
3. **fg_color**: The foreground color of the text box.
4. **bg_color**: The background color of the text box.
5. **border_color**: The color of the text box's border.
6. **border_width**: The width of the text box's border.
7. **corner_radius**: The radius of the text box's corners.
8. **font**: The font used for the text in the text box.
9. **text_color**: The color of the text in the text box.
10. **state**: The state of the text box (e.g., 'normal', 'disabled').
11. **wrap**: The wrap mode of the text box (e.g., 'none', 'char', 'word').
12. **insertbackground**: The color of the insertion cursor.
13. **insertwidth**: The width of the insertion cursor.
14. **selectbackground**: The background color of the selected text.
15. **selectforeground**: The foreground color of the selected text.
16. **yscrollcommand**: Associates a vertical scrollbar with the text box.
17. **xscrollcommand**: Associates a horizontal scrollbar with the text box.

You can configure these attributes using the `configure` method or directly when creating the text box. Here's an example:

```
import customtkinter as ctk

root = ctk.CTk()

text_box = ctk.CTkTextbox(
    master=root,
    width=400,
    height=200,
    fg_color="white",
    bg_color="gray",
    border_color="black",
    border_width=2,
    corner_radius=10,
    font=("Arial", 14),
    text_color="black",
    state="normal",
    wrap="word",
    insertbackground="black",
    insertwidth=2,
    selectbackground="blue",
    selectforeground="white"
)

text_box.pack(pady=20)

root.mainloop()
```

To create input popups in CustomTkinter, you can use the `CTkInputDialog` widget. Here are the attributes you can use with this widget:

1. **title**: The title of the popup window.
2. **text**: The message displayed in the popup window.
3. **button_text**: The text displayed on the confirmation button.
4. **width**: The width of the popup window.
5. **height**: The height of the popup window.
6. **fg_color**: The foreground color of the popup window.
7. **bg_color**: The background color of the popup window.
8. **border_color**: The color of the popup window's border.
9. **border_width**: The width of the popup window's border.
10. **corner_radius**: The radius of the popup window's corners.
11. **font**: The font used for the text in the popup window.
12. **text_color**: The color of the text in the popup window.
13. **button_color**: The color of the confirmation button.
14. **button_hover_color**: The hover color of the confirmation button.

Here's an example of how to create an input popup:

```
import customtkinter as ctk

def get_input():
    dialog = ctk.CTkInputDialog(
        title="Input Dialog",
        text="Please enter your name:",
        button_text="Submit",
        width=300,
        height=150,
        fg_color="white",
        bg_color="gray",
        border_color="black",
        border_width=2,
        corner_radius=10,
        font=("Arial", 14),
        text_color="black",
        button_color="blue",
        button_hover_color="lightblue"
    )
    user_input = dialog.get_input()
    print("User input:", user_input)

root = ctk.CTk()

button = ctk.CTkButton(
    master=root,
    text="Open Input Dialog",
    command=get_input
)

button.pack(pady=20)

root.mainloop()
```

This example creates a button that, when clicked, opens an input dialog where the user can enter their name. The input is then printed to the console.

Here are the attributes you can use with the `CTkOptionMenu` widget in CustomTkinter:

1. **values**: A list of values to display in the option menu.
2. **variable**: Associates a Tkinter variable (e.g., `StringVar`) with the option menu.
3. **command**: The function to be called when an option is selected.
4. **width**: The width of the option menu.
5. **height**: The height of the option menu.
6. **fg_color**: The foreground color of the option menu.
7. **bg_color**: The background color of the option menu.
8. **border_color**: The color of the option menu's border.
9. **border_width**: The width of the option menu's border.
10. **corner_radius**: The radius of the option menu's corners.
11. **font**: The font used for the text in the option menu.
12. **text_color**: The color of the text in the option menu.
13. **state**: The state of the option menu (e.g., 'normal', 'disabled').

You can configure these attributes using the `configure` method or directly when creating the option menu. Here's an example:

```
import customtkinter as ctk

def option_selected(choice):
    print("Selected option:", choice)

root = ctk.CTk()

option_var = ctk.StringVar(value="Option 1")

option_menu = ctk.CTkOptionMenu(
    master=root,
    values=["Option 1", "Option 2", "Option 3"],
    variable=option_var,
    command=option_selected,
    width=200,
    height=30,
    fg_color="white",
    bg_color="gray",
    border_color="black",
    border_width=2,
    corner_radius=10,
    font=("Arial", 14),
    text_color="black",
    state="normal"
)
option_menu.pack(pady=20)

root.mainloop()
```

This example creates an option menu with three options and prints the selected option to the console when an option is chosen.