Python TUTORIAL

# Tkinter Command Binding

**Summary**: in this tutorial, you'll learn about the Tkinter command binding that associates a callback with an event of a widget.

## Introduction to Tkinter command binding

To make the application more interactive, the widgets need to respond to the events such as:

- Mouse clicks
- Key presses

This requires assigning a callback function to a specific event. When the event occurs, the callback will be invoked automatically to handle the event.

In Tkinter, some widgets allow you to associate a callback function with an event using the command binding.

It means that you can assign the name of a function to the command option of the widget so that when the event occurs on the widget, the function will be called automatically.

To use the command binding, you follow these steps:

- First, define a function as a callback.

- Then, assign the name of the function to the `command` option of the widget.

For example, the following defines a function called `button_clicked()` :

```
def button_clicked():
    print('Button clicked')
```

After that, you can associate the function with the `command` option of a button widget (https://www.pythontutorial.net/tkinter/tkinter-button/) :

```
ttk.Button(root, text='Click Me',command=button_clicked)
```

Note that you pass the callback without parentheses `()` within the `command` option. Otherwise, the callback would be called as soon as the program runs.

The following is the full program that illustrates how to associate the `button_clicked` callback function with the `Button` widget:

```
import tkinter as tk
from tkinter import ttk

root = tk.Tk()


def button_clicked():
    print('Button clicked')


button = ttk.Button(root, text='Click Me', command=button_clicked)
button.pack()

root.mainloop()
```

## Tkinter button command arguments

If you want to pass arguments to a callback function, you can use a lambda expression (https://www.pythontutorial.net/python-basics/python-lambda-expressions/) .

First, define a function that accepts arguments:

```
def callback_function(args):
    # do something
```

Then, define a lambda expression and assign it to the `command` option. Inside the lambda expression, invoke the callback function:

```
ttk.Button(
    root,
    text='Button',
    command=lambda: callback(args))
```

The following program illustrates how to pass an argument to the callback function associated with the button command:

```
import tkinter as tk
from tkinter import ttk

root = tk.Tk()


def select(option):
    print(option)


ttk.Button(root, text='Rock', command=lambda: select('Rock')).pack()
ttk.Button(root, text='Paper',command=lambda: select('Paper')).pack()
ttk.Button(root, text='Scissors', command=lambda: select('Scissors')).pack()

root.mainloop()
```

When you click a button, the lamda expression bound to the command of that button will execute. It'll call the `select()` function and pass a string argument to it.

## Limitations of command binding

First, the `command` option isn't available in all widgets. It's limited to the `Button` and some other widgets.

Second, the `command` button binds to the left-click and the backspace. It doesn't bind to the `Return` key.

To check this you can move focus to a button in the program above and press the backspace and return keys. This is not really user-friendly. Unfortunately, you cannot change the binding of the `command` function easily.

To overcome these limitations, Tkinter provides an alternative way for associating a function with an event, which is called event binding (https://www.pythontutorial.net/tkinter/tkinter-event-binding/) .

## Summary

- Assign a function name to the `command` option of a widget is called command binding in Tkinter.
- The assigned function will be invoked automatically when the corresponding event occurs on the widget.
- Only few widgets support the `command` option.