How to Schedule an Action with Tkinter after() method

Summary: in this tutorial, you'll learn how to use the Tkinter `after()` method to schedule an action after a timeout has elapsed.

Introduction to Tkinter after() method

All Tkinter widgets have the `after()` method with the following syntax:

```
widget.after(delay, callback=None)
```

The `after()` method calls the `callback` function once after a `delay` milliseconds (ms) within Tkinter's main loop.

If you don't provide the `callback` , the `after()` method behaves like the `time.sleep()` function. However, the `after()` method uses the millisecond instead of the second as the unit.

Tkinter after() method example

Let's see the following program:

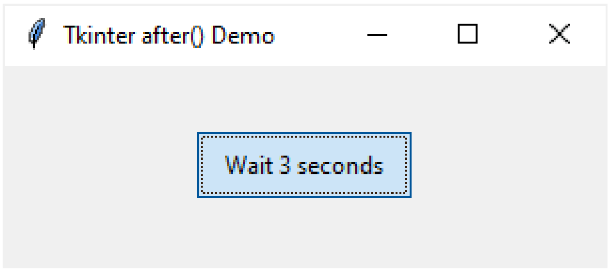
```
import tkinter as tk
from tkinter import ttk
import time


class App(tk.Tk):
    def __init__(self):
        super().__init__()

        self.title('Tkinter after() Demo')
        self.geometry('300x100')

        self.style = ttk.Style(self)

        self.button = ttk.Button(self, text='Wait 3 seconds')
        self.button['command'] = self.start
        self.button.pack(expand=True, ipadx=10, ipady=5)

    def start(self):
        self.change_button_color('red')
        time.sleep(3)
        self.change_button_color('black')

    def change_button_color(self, color):
        self.style.configure('TButton', foreground=color)


if __name__ == "__main__":
    app = App()
    app.mainloop()
```

The program consists of a button. When you click the button:

- First, the color of the button turns red.
- Then, the program sleeps for 3 seconds.

- Finally, the color of the button turns black.

However, when you run the program and click the button, you'll notice that the color of the button doesn't change at all. Also, the window freezes for 3 seconds like this:

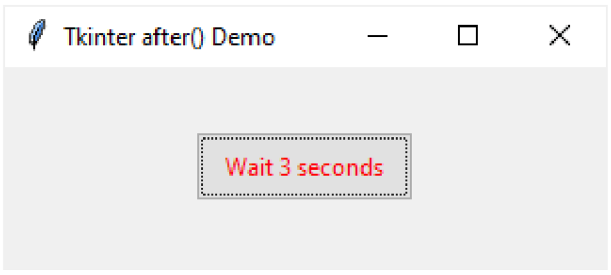


The reason was that the `sleep()` function suspended the main thread execution. Therefore, Tkinter couldn't update the GUI.

To fix the issue, you can use the `after()` method to schedule the action that updates the color of the button instead of suspending the main thread execution. For example:

```
import tkinter as tk
from tkinter import ttk
import time


class App(tk.Tk):
    def __init__(self):
        super().__init__()

        self.title('Tkinter after() Demo')
        self.geometry('300x100')

        self.style = ttk.Style(self)

        self.button = ttk.Button(self, text='Wait 3 seconds')
        self.button['command'] = self.start
        self.button.pack(expand=True, ipadx=10, ipady=5)

    def start(self):
        self.change_button_color('red')
        self.after(3000,lambda: self.change_button_color('black'))
    def change_button_color(self, color):
        self.style.configure('TButton', foreground=color)
        print(color)


if __name__ == "__main__":
    app = App()
    app.mainloop()
```

Output:



Why use Tkinter after() method

A Python program can have one or multiple threads (https://www.pythontutorial.net/advanced-python/python-threading/) . When you launch a Tkinter application, it executes in the main thread.

The Tkinter's main loop must start from the main thread. It's responsible for handling events and updating GUI.

If you start a long-running task in the main thread, the GUI will freeze and don't respond to user events.

To prevent a long-running task from blocking the main thread, you can schedule an action that won't be executed earlier than a specified time by using the `after()` method.

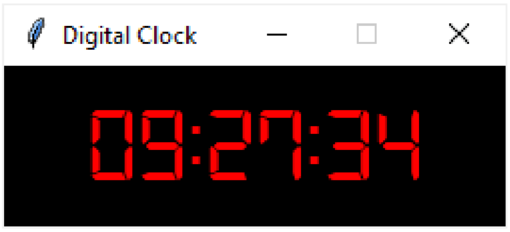
Tkinter will execute the callback in the main thread when the main thread is **not busy**.

A practical Tkinter after() method example

The following program displays a digital clock. It uses the `after()` method to update the current time every second:

```
import tkinter as tk
from tkinter import ttk
import time


class DigitalClock(tk.Tk):
    def __init__(self):
        super().__init__()

        # configure the root window
        self.title('Digital Clock')
        self.resizable(0, 0)
        self.geometry('250x80')
        self['bg'] = 'black'

        # change the background color to black
        self.style = ttk.Style(self)
        self.style.configure(
            'TLabel',
            background='black',
            foreground='red')

        # label
        self.label = ttk.Label(
            self,
            text=self.time_string(),
            font=('Digital-7', 40))

        self.label.pack(expand=True)

        # schedule an update every 1 second
        self.label.after(1000, self.update)
    def time_string(self):
        return time.strftime('%H:%M:%S')

    def update(self):
        """ update the label every 1 second """

        self.label.configure(text=self.time_string())

        # schedule another timer
        self.label.after(1000, self.update)


if __name__ == "__main__":
    clock = DigitalClock()
    clock.mainloop()
```

Output:



How it works.

The following method returns the current time in the string format:

```
def time_string(self):
    return time.strftime('%H:%M:%S')
```

The `__init__()` method uses the `after()` method to schedule an action that updates the current time to the label every second:

```
self.label.after(1000, self.update)
```

In the `update()` method, update the current time to the label, and schedule another update after one second:

```
def update(self):
    """ update the label every 1 second """

    self.label.configure(text=self.time_string())

    # schedule another timer
    self.label.after(1000, self.update)
```

Note that this program uses the Digital 7 font from the 1001fonts.com (https://www.1001fonts.com/digital-7-font.html)

Summary

- Use the Tkinter `after()` method to schedule an action that will run after a timeout has elapsed
- The callback passed into the `after()` method still runs in the main thread. Therefore, you should avoid performing the long-running task using the `after()` method.