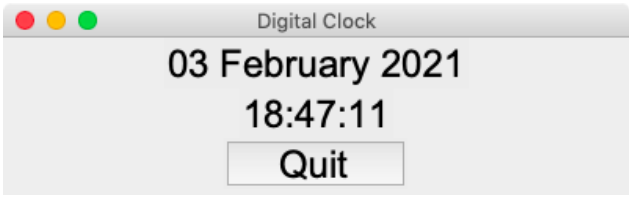
Lab 4: GUI and Event-Driven Programming

01219116/117 Computer Programming II

In this lab assignment, we will create a simple digital clock application using Tkinter. Our application will look similar to the screenshot below.



Here is the detailed specification:

- The dimension of the application's window is 400x100 pixels.
- The application window is displayed with the title **Digital Clock**.
- The first line is a label widget displaying the current date in the form of **<day> <month> <year>**. The date must be updated when the application runs past midnight.
- The second line is another label widget displaying the current time in the form of **<hour>:<minute>:<second>**, which gets updated every second.
- There is also a Quit button that, once clicked, will close the application.
- All widgets' texts are displayed in the Arial font with the font size of 24 points.

Now proceed with the following steps.

1. Define the Application class that creates a root window. Use the following code as your starting point.

```
import tkinter as tk
import tkinter.ttk as ttk

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

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

2. Inside the `__init__()` method, add more code to set the application's title and window size as described in the specification. In addition, place two label widgets displaying static texts as placeholders, and a Quit button, all appearing in the specified font as shown. (The Quit button may still do nothing at this point.)

Put the added code in the empty box below (adjust the box to fit all your code; also use the **Paste without formatting** menu to avoid messing up the code box):

```
import tkinter as tk
import tkinter.ttk as ttk


class Application(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("Digital Clock")
        self.geometry("400x100")
        self.style = ttk.Style()
        self.style.configure(".", font=("Arial", 24))
        self.date = ttk.Label(text="date")
        self.date.pack()
        self.time = ttk.Label(text="time")
        self.time.pack()
        self.button = ttk.Button(text="Quit")
        self.button.pack()

    def run(self):
        self.mainloop()


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

3. Bind the Quit button's click event to a function that closes the application when clicked. Put the line of code that defines this behaviour in the box below. (In case you have already done so in the previous step, just put a comment saying so.)

```
def quit(self):
    self.destroy()
```

4. Define two control variables of type string as instance members, named `str_date` and `str_time`, to store the current date and time, respectively. Bind both variables to the two label widgets created earlier. The labels must display the current values of these two variables as soon as they are updated. For example, if the variables are updated by the statements:

```
self.str_date.set("*** Current Date ***")
self.str_time.set("*** Current Time ***")
```

then the application window should appear like this

Put all related code in the box.

```
self.str_date = tk.StringVar()
self.str_time = tk.StringVar()
self.date = ttk.Label(textvariable=self.str_date)
self.time = ttk.Label(textvariable=self.str_time)
self.str_date.set("*** Current Date ***")
self.str_time.set("*** Current Time ***")
```

5. Define the `update_time()` method that updates the variables `str_date` and `str_time` to the current date and time, respectively, with the format described in the specification. This method must also schedule an event to call itself one second in the future.
Hint: study the `now()` and `strftime()` functions provided by the `datetime` module.

Put the method definition in the box.

```
    def update_time(self):
        date_now = datetime.now().strftime("%d %B %Y")
        time_now = datetime.now().strftime("%H:%M:%S")
        self.str_date.set(date_now)
        self.str_time.set(time_now)
        self.after(1000, self.update_time)
```

6. Add a line of code to call the `update_time()` method once at the end of the `__init__()` method. Your application should now work as expected.

Complete Code

Place the complete code of your application in the box below. (Adjust the box to fit all the code.)

```
import tkinter as tk
import tkinter.ttk as ttk
from datetime import datetime

class Application(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("Digital Clock")
        self.geometry("400x100")
        self.style = ttk.Style()
        self.style.configure(".", font=("Arial", 24))
        self.str_date = tk.StringVar()
        self.str_time = tk.StringVar()
        self.date = ttk.Label(textvariable=self.str_date)
        self.time = ttk.Label(textvariable=self.str_time)
        self.update_time()
        self.date.pack()
        self.time.pack()
        self.button = ttk.Button(text="Quit", command=self.quit)
        self.button.pack()

    def update_time(self):
        date_now = datetime.now().strftime("%d %B %Y")
        time_now = datetime.now().strftime("%H:%M:%S")
        self.str_date.set(date_now)
        self.str_time.set(time_now)
        self.after(1000, self.update_time)

    def quit(self):
        self.destroy()

    def run(self):
        self.mainloop()

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