CS3010 – Lab 3 Exercise – Basic GUI

The objective of this exercise is to practice basic GUI development in Python using Tkinter and to get a practical understanding of the GUI execution thread in a GUI main loop using a simple GUI program.

Task 1- Creating a new counter

Despite this being the first of the GUI exercise this task does not involve any graphical interface but is focused on user interaction in general and forms the foundation for the second task. In this tutorial, you are asked to create a timer program that uses the command line. In the following tutorials, we will build different types of GUIs for these programs.
The datetime library module provides methods for manipulating objects representing dates and times. To use this library you must put:

```
from datetime import datetime
```

at the start of your program.

The method `datetime.now()` returns the current date and time of the system. You can use attributes hour, minute, and second to access details of the current time.

a) You can use the current system time to measure the elapsed time during the execution of a program. Write a reaction timer program that finds the start time and store it in a variable. The program then prompts the user to press the "return" key, inputs an empty string from the keyboard when the user has pressed "return", finds the current time and the difference between the current time and the start time. It then prints the difference time in seconds.

Hint: you can use the `total_seconds()` method on a time difference to get the time difference in seconds as a float number.

b) Consider the following function that pauses for a specified time in milliseconds:

```
def pause(delay):
     start = datetime.now()
while True:
     diff = (datetime.now() - start)
     diff = diff.total_seconds()*1000 #convert to milliseconds
     if (diff > delay):
     break
```

Write a timer program that prompts for and inputs a time in seconds and then, once a second, prints the time number and decrements it until it is 0.

c) The technique used in part b to pause execution is called "Busy Waiting". While being easy to implement it comes with certain disadvantages. Try finding out what these are by googling or having a look at your CPU load while the timer program is running. Modify your code in part b to use the `sleep()` method of the `time` module instead of your pause method.

d)

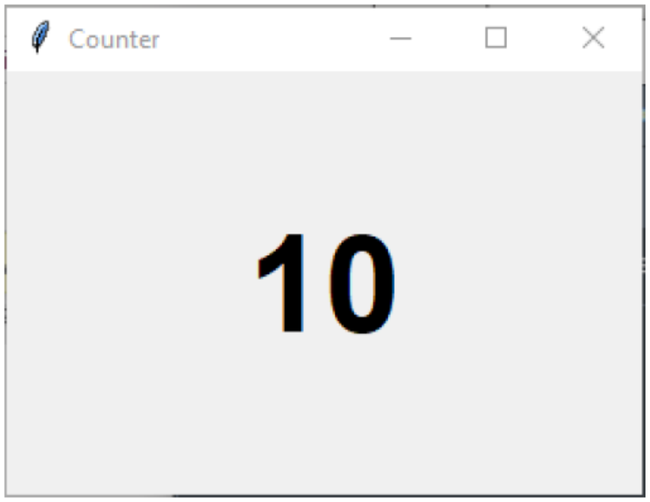
```
import time
...
time.sleep(3.5) # wait for 3.5 seconds
```

Is `sleep()` also a busy waiting pause?

Task 2 – Creating a GUI interface for the counter program

In this task, you are going to add a simple GUI to the counter code developed in the previous task.

Create a simple GUI for task1-c using a single Label to show the counter value. An example is shown below:



Add the required code to display the initial value of the counter in the GUI as well as printing the value of the counter in the console. Write the required code to update the value in the GUI display every second.

You will notice that the GUI will not update as the program prints the countdown values. It only updates the GUI once the counter has reached zero.

Task 3 – Making the counter GUI responsive

The reason for the GUI not being updated in **task 2** is that the GUI has its execution thread which starts when you call the mainloop()method. You will only see the GUI window when the `mainloop()` method is called. After calling the `mainloop()` method the rest of the code after this call will not execute and all the interactions will be managed in the main loop of the GUI.

If you would like to perform a task in a certain time interval in the main loop of a GUI you can use the `after()` method.

```
root.after(delay, function)
```

The method takes a delay parameter in milliseconds and a function that will be called after the delay amount of time is passed. To create a repeated call of the function you can call the after method again in the function itself.

Modify your program developed in task2 to use the `after()` method to update the display every second.