Project Report

On

"Digital clock & stopwatch using Python"

Table of Contents

Introduction

Python:

Python is a high-level, interpreted programming language that is widely used for web development, scientific computing, data analysis, artificial intelligence, and more. It was first released in 1991 by Guido van Rossum and has since become one of the most popular programming languages in the world.

One of the key features of Python is its readability and simplicity. The language is designed to be easy to read and understand, making it a great choice for beginners and experienced programmers alike. Python uses indentation to indicate code blocks, rather than curly braces or other symbols, which helps to make the code more readable.

Python also has a large and active community that has developed a wide range of libraries and modules that can be used to extend the capabilities of the language. Some of the most popular libraries include NumPy, Pandas, and Matplotlib for data analysis, and TensorFlow and PyTorch for machine learning.

Python is also an interpreted language, which means that it does not need to be compiled before it is executed. This can make the development process faster and more efficient, as changes can be made and tested without the need for a separate compilation step.

In addition to its general-purpose nature, Python is also often used for specific tasks such as web development, scientific computing, and data analysis. For web development, Python offers frameworks like Django and Flask which makes it easy to create web applications. In scientific computing, libraries like NumPy and SciPy provide advanced mathematical and scientific capabilities,

while libraries like Pandas make it easy to work with large datasets in data analysis.

Python also has a wide range of applications in artificial intelligence, machine learning, and deep learning. TensorFlow, Keras and PyTorch are popular libraries that are widely used in these fields to build and train neural networks.

In conclusion, Python is a powerful and versatile programming language that is well-suited to a wide range of tasks. Its readability and simplicity make it a great choice for beginners and experienced programmers alike, while its large and active community has developed a wide range of libraries and modules that can be used to extend its capabilities. Python's wide range of applications, from web development to artificial intelligence, makes it a popular choice for many types of projects.

Why to make digital clock and stopwatch using Python:

There are several reasons why you might want to create a clock and stopwatch using Python:

1. **Learning and practice:** Creating a clock and stopwatch using Python can be a great way to learn and practice programming concepts such as loops, conditionals, and functions. It can also help you to become more familiar with Python's built-in modules and libraries, such as the time module.
2. **Customization:** By creating your own clock and stopwatch using Python, you can customize the functionality and appearance to suit your specific needs. For example, you could create a stopwatch that saves lap times, or a clock that displays the date and time in different formats.
3. **Reusability:** Once you have created a clock and stopwatch using Python, you can easily reuse the code in other projects. This can save you time and effort compared to creating a new clock and stopwatch from scratch each time.
4. **Integration with other projects:** A clock and stopwatch created using Python can be integrated into other projects such as a time tracking app, a fitness app or a personal project management tool.
5. **Personal or Professional use**: you could use your clock and stopwatch code in your personal projects or use it as a tool in your professional projects like in a manufacturing unit to measure the time taken for a specific task or in a laboratory to measure reaction time.
6. Overall, creating a clock and stopwatch using Python can be a fun and educational project that can also be useful in a variety of ways. It can help you to improve your programming skills, create a useful tool for your personal or professional use and can be integrated in other projects.

Begin with the project

The amazing part of creating our Graphical User Interface (GUI) applications is that we can customize them the way we want. There are various features available for customization ranging from the font of the text to the background colour.

In order to build the Digital Clock and Stopwatch with GUI in Python, we will need the following modules:

1. **Tkinter:** Tkinter is the standard GUI library for Python. Python when combined with Tkinter provides a fast and easy way to create GUI applications. Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit.
2. **Time:** The **time** module will allow us to work with time.

Since both these modules come pre-installed with Python, there is no need for us to install them separately. We just need to import both of these modules.

The procedure of building the Digital Clock and Stopwatch in Python is divided into several steps for better understanding. The steps we will need to execute are as follows:

Step 1: We will start by importing the required modules.

Step 2: We will then define the function to display current time.

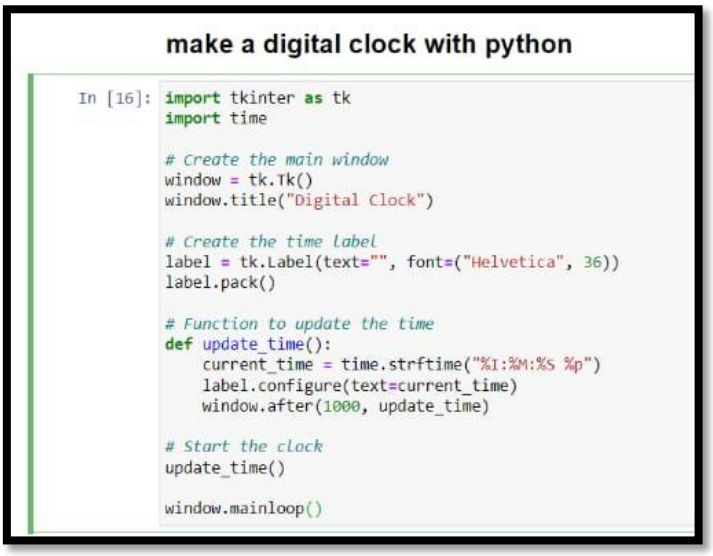
Step 3: We will then create a main window for the application.

Step 4: We will add widgets to the window.

At first, it is important to import the required modules which are necessary modules to help us build this project. These modules include the **tkinter** module to add a graphical user interface to the application and the **time** module to work with dates.

We will define a function that will allow us to display the current time whenever the program is executed. This function will use the **strttime()** method of the **tkinter** module to represent the current time in the string format. We will then convert the 24-hour time to 12-hour time and set the value as AM or PM in the label per the requirement. We will also insert the evaluated values in the labels we create later.

Digital clock: Python code



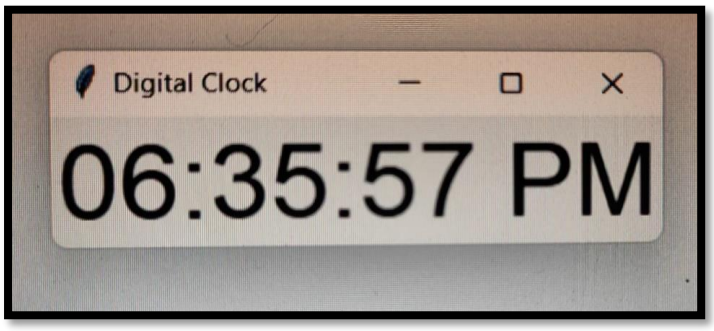
The above code creates a digital clock using the Python programming language and the Tkinter library for the graphical user interface (GUI).

Here's a breakdown of the code:

1. The first few lines import the necessary modules: time, from tkinter import *
2. The Tk() function creates a Tkinter window and assigns it to the variable "root". The title() method sets the title of the window to "Digital Clock".
3. The time() function retrieves the current time, formats it as a string (hours, minutes, and seconds), and updates the text of the label. The function also uses the after() method to schedule itself to be called again in 1,000 milliseconds (1 second), which causes the clock to update every second.
4. The Label() function creates a label widget and assigns it to the variable "lbl". The font, background, and foreground options are set to customize the appearance of the label.
5. The pack() method is used to place the label in the canter of the window.
6. The mainloop() function starts the Tkinter event loop, which allows the clock to update and the window to respond to user input.

Output:

When we run above code in python, we will get out digital clock. Here is the output of the above code. The digital clock will show the time according to the time in your personal computer. This clock will always on your system after you run the code in python.



Stopwatch: Python code

```
In [20]: # Python program to illustrate a stop watch
         # using Tkinter
         #importing the required libraries
         import tkinter as Tkinter
         from datetime import datetime
         counter = 66600
         running = False
         def counter_label(label):
             def count():
                 if running:
                     global counter

                     # To manage the initial delay.
                     if counter==66600:
                         display="Starting..."
                     else:
                         tt = datetime.fromtimestamp(counter)
                         string = tt.strftime("%H:%M:%S")
                         display=string

                     label['text']=display # Or label.config(text=display)

                     # label.after(arg1, arg2) delays by
                     # first argument given in milliseconds
                     # and then calls the function given as second argument.
                     # Generally like here we need to call the
                     # function in which it is present repeatedly.
                     # Delays by 1000ms=1 seconds and call count again.
                     label.after(1000, count)
                     counter += 1
```

We will start the project by using preinstalled libraries which are essential to run our code smoothly.

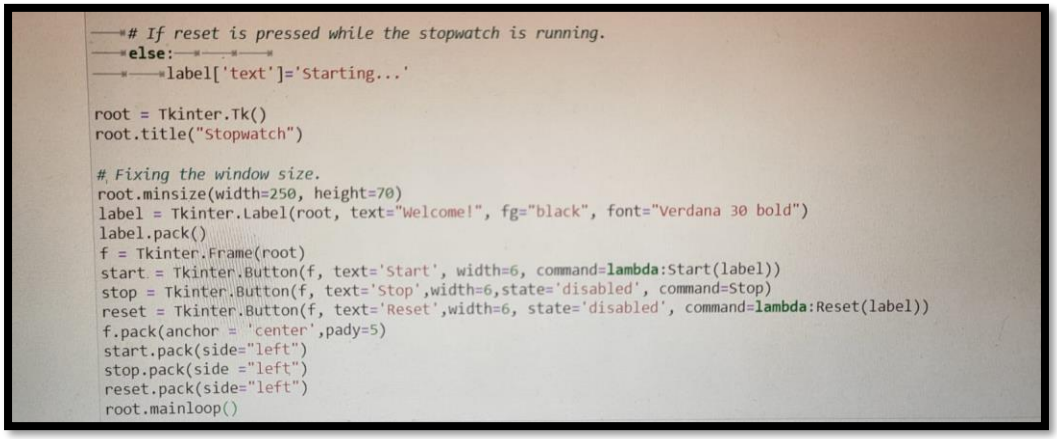
```
    # Triggering the start of the counter.
    count()

# start function of the stopwatch
def Start(label):
    global running
    running=True
    counter_label(label)
    start['state']='disabled'
    stop['state']='normal'
    reset['state']='normal'

# Stop function of the stopwatch
def Stop():
    global running
    start['state']='normal'
    stop['state']='disabled'
    reset['state']='normal'
    running = False

# Reset function of the stopwatch
def Reset(label):
    global counter
    counter=66600

    # If rest is pressed after pressing stop.
    if running==False:
        reset['state']='disabled'
        label['text']='Welcome!'
```



```
    # If reset is pressed while the stopwatch is running.
    else:
        label['text']='Starting...'

root = Tkinter.Tk()
root.title("Stopwatch")

# Fixing the window size.
root.minsize(width=250, height=70)
label = Tkinter.Label(root, text="Welcome!", fg="black", font="Verdana 30 bold")
label.pack()
f = Tkinter.Frame(root)
start = Tkinter.Button(f, text='Start', width=6, command=lambda:Start(label))
stop = Tkinter.Button(f, text='Stop',width=6,state='disabled', command=Stop)
reset = Tkinter.Button(f, text='Reset',width=6, state='disabled', command=lambda:Reset(label))
f.pack(anchor = 'center',pady=5)
start.pack(side="left")
stop.pack(side ="left")
reset.pack(side="left")
root.mainloop()
```

A stopwatch is a handheld timepiece designed to measure the amount of time elapsed from a particular time when it is activated to the time when the piece is deactivated. A large digital version of a stopwatch designed for viewing at a distance, as in a sports stadium, is called a stop clock. In manual timing, the clock is started and stopped by a person pressing a button. In fully automatic time, both starting and stopping are triggered automatically, by sensors.

Here's a breakdown of the code:

1. The above code defines a **Digital Stopwatch** class that creates a graphical user interface (GUI) using the Tkinter library. The GUI has a label that displays the elapsed time, and buttons for starting, stopping, and pausing the stopwatch.

2. The **__init__** method creates the root window, sets its title, creates a label to display the time, and calls the **create_buttons** and **update_time** methods to create the buttons and start the timer.
3. The **create_buttons** method creates the start, stop, and pause buttons and sets their command properties to call the **start**, **stop**, and **restart** methods respectively. It also sets the initial state of the buttons, so that the stop and pause buttons are disabled until the stopwatch is started.

4. The **start**, **stop**, and **restart** methods are responsible for controlling the stopwatch's state and updating the GUI accordingly.

5. When the start button is pressed, the start method sets the start_time variable to the current time, sets running to true and enables the stop and pause button and disables the start button.

6. When the stop button is pressed, the stop method sets the end_time variable to the current time, sets running to false and enables the start button and disables the stop and pause button.

7. When the pause button is pressed, the pause method sets the restart_start_time variable to the current time, sets paused to true and enables the start button and disables the pause and stop button.

8. The **get_elapsed_time** method calculates the elapsed time since the stopwatch was started by subtracting the start_time and restart_time from the current time.

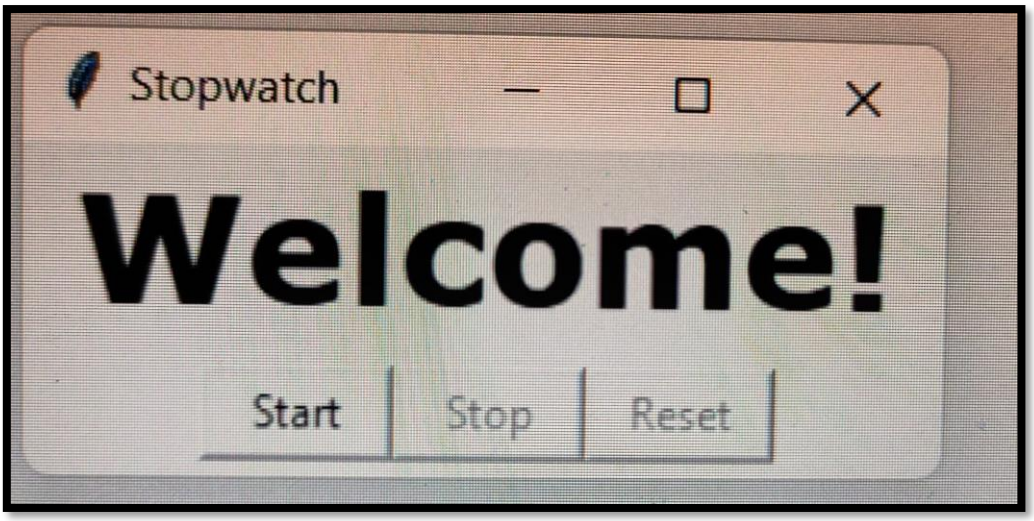
9. The **update_time** method updates the label with the current elapsed time, formatted as minutes and seconds. It uses the **divmod** function to split the elapsed time into minutes and seconds, and then formats the string to display leading zeroes for single-digit minutes and seconds. It also uses the **root.after** method to call itself repeatedly, so that the time is updated every second as long as the stopwatch is running.

10. Finally, the last line creates an instance of the Digital Stopwatch class and starts the Tkinter event loop.

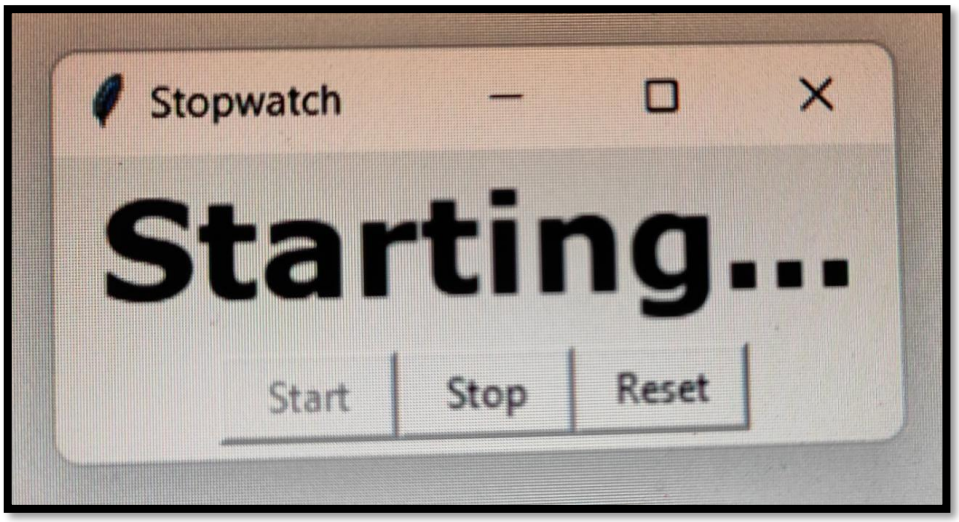
11. You can test this by running the code and clicking on the start, pause and stop button, the label should display the elapsed time.

Output:

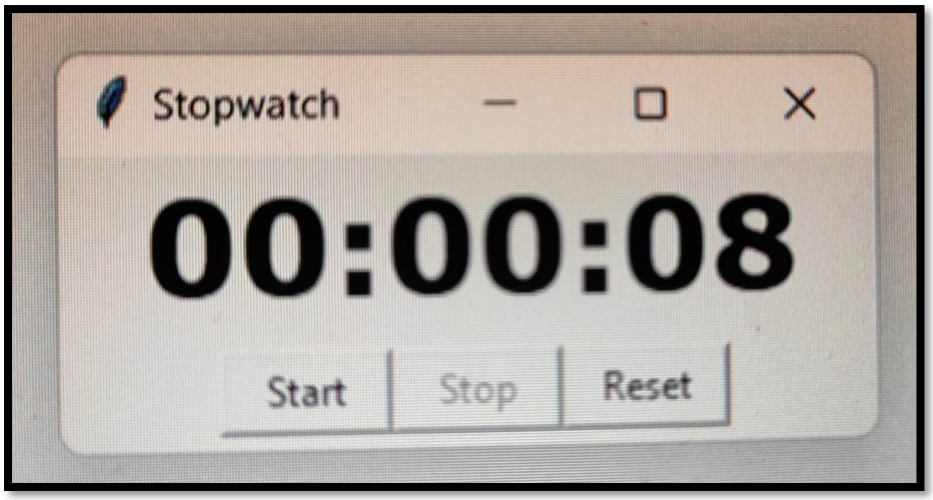
When we run above code in python, we will get our stopwatch. Here is the output of the above code. This stopwatch will always on your system after you run the code in python. Once you run your code, “welcome” will show until your press start button.



Once you click on start button, “Starting” will show, it means your stopwatch is about to start.



Once you start to see the time, you can stop whenever you desire to stop the time. The moment you press “stop”; your stopwatch will automatically stop. You can also reset the timer by using “Reset” button. And you will see your watch from the very beginning. You can again use “Start” button to use it again.



Conclusion

A digital clock and stopwatch are convenient tool for measuring time in a variety of settings. The clock can be used to tell the time at a glance, while the stopwatch can be used for timing events and activities.

This project aims to create a digital clock and stopwatch using the Python programming language and the Tkinter library for creating a graphical user interface (GUI). The clock will display the current time in a digital format, and the stopwatch will allow the user to start, stop, and pause the timer.

The project begins by importing the necessary libraries, including Tkinter for creating the GUI, and time for measuring time. The project then defines a Digital Clock class that creates a window with a

label to display the time. The class includes an update_time method that updates the label with the current time and calls itself repeatedly using the root.after method.

The project also defines a Digital Stopwatch class that creates a window with a label to display the elapsed time, and buttons for starting, stopping, and reset the timer. The class includes methods for controlling the stopwatch's state and updating the label with the elapsed time.

The two classes are independent and can be run separately, but they can also be combined in a single program to create a clock and stopwatch in the same window.

The user interface of the digital clock and stopwatch is simple and easy to use. The clock displays the current time and updates it every second. The stopwatch allows the user to start, stop, and reset the timer, and displays the elapsed time when stopped.

Overall, this project demonstrates how to create a functional digital clock and stopwatch using Python and Tkinter. The resulting program is a useful tool for measuring time in a variety of settings, and can be easily customized to suit different needs.

Bibliography

- https://www.javatpoint.com/
- https://www.geeksforgeeks.org/
- https://www.w3schools.com/