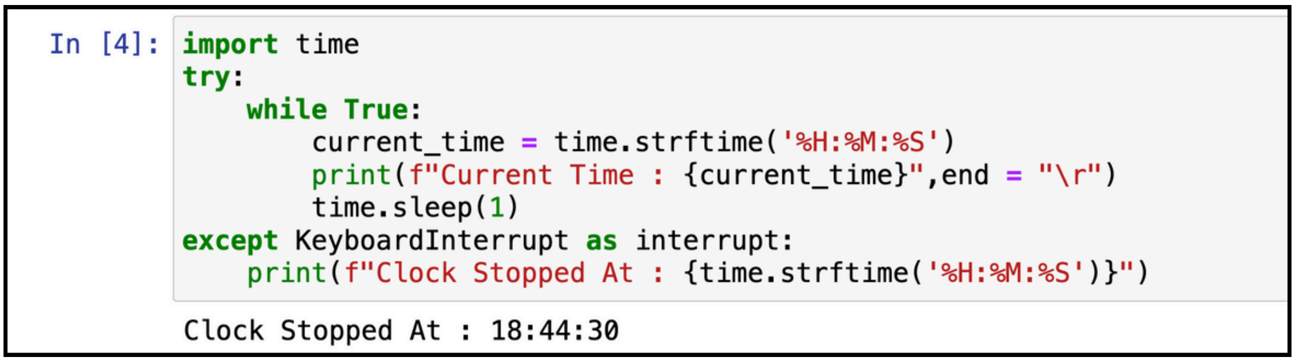
learnwithbhawana

Python Project Day 4 - Real-Time Clock ⏱️

In this project, we create a Real-Time Clock that continuously updates the current time in the Python console using the **time module** and a **carriage return** to refresh the same line.

Code 💻 :

```
import time
try:
    while True:
        current_time = time.strftime('%H:%M:%S')
        print(f"Current Time : {current_time}",end = "\r")
        time.sleep(1)
except KeyboardInterrupt as interrupt:
    print(f"Clock Stopped At : {time.strftime('%H:%M:%S')}")
```



```
In [4]: import time
        try:
            while True:
                current_time = time.strftime('%H:%M:%S')
                print(f"Current Time : {current_time}",end = "\r")
                time.sleep(1)
        except KeyboardInterrupt as interrupt:
            print(f"Clock Stopped At : {time.strftime('%H:%M:%S')}")

        Clock Stopped At : 18:44:30
```

Line-by-Line Explanation 🔍 :

1. **import time**:

   Imports Python's time module to access the current system time.

2. **try:**

Starts a try block to handle any keyboard interruption (like Ctrl + C).

3. **while True:**

   Runs an infinite loop to keep the clock running continuously.
4. **current_time = time.strftime('%H:%M:%S'):**

   Gets the current time in hours:minutes:seconds format.
5. **print(f"Current Time : {current_time}", end = "\r"):**

   Prints the current time and uses carriage return '\r' to refresh the same line instead of moving to the next line.
6. **time.sleep(1):**

   Waits for 1 second before updating the time again.
7. **except KeyboardInterrupt:**

   Stops the clock gracefully when the user presses Ctrl + C.
8. **print(f"Clock Stopped At : {time.strftime('%H:%M:%S')}"):**

   Prints the final time when the clock was stopped.

Why Carriage Return (\r) is Used:

1. Carriage return **'\r'** helps in refreshing the same line in the console output.
2. It prevents printing on a new line each second, making the clock appear to run in real-time at one place.

Applications of This Code:

1. Real-time clocks in terminal-based apps.
2. Countdown timers.
3. Real-time progress bars.
4. Live status displays (like download or system monitoring).
5. Digital stopwatches.

What Else Can We Try?

1. Add date with time. (time.strftime('%d/%m/%Y %H:%M:%S'))
2. Create a countdown timer instead of a clock.

```
import time

timer_limit = int(input("Enter countdown time in seconds"))

try:
   while timer_limit:
      print(f"Timer : {timer_limit} seconds remaining. ")
      timer_limit -= 1
      time.sleep(1)
   print("Times up !!")
except KeyboardInterrupt as e:
   print(f"\n Timer stopped manually at {timer_limit} seconds")
```

3. Build a GUI clock using Tkinter.*(I will create separate project for the same)*
4. Add start/stop buttons using GUI.*(I will create separate project for the same)*
5. Make an alarm clock with sound notifications.(Use winsound,refer link: winsound — Sound-playing interface for Windows)

THE END