ChatGPT

Applicazione di dattilografia in Python con Tkinter

Requisiti principali: l'applicazione deve leggere un testo da un file modificabile, mostrarlo a video e fornire un campo di input. L'utente deve digitare il testo esattamente carattere per carattere: ogni errore blocca l'avanzamento (non viene accettato nel campo) e aggiunge 0,5 secondi di penalità. Il cronometro parte con il primo carattere corretto e si ferma al termine del testo; alla fine viene mostrato il tempo complessivo (tempo reale + penalità).

Lettura del testo da file

Per caricare il testo da digitare utilizziamo le funzioni base di I/O di Python. Ad esempio, il testo può essere salvato in un file `testo.txt` e letto con il costrutto `open()` e il metodo `read()` [1]:

```
with open("testo.txt", "r", encoding="utf-8") as f:
    testo = f.read().strip()
```

In questo modo la stringa completa viene caricata in memoria e poi mostrata nella GUI.

Interfaccia grafica con Tkinter

Per la GUI usiamo **Tkinter**, la libreria standard di Python per interfacce grafiche. Tkinter è uno strumento ampiamente utilizzato e permette di creare finestre in modo semplice [2]. Creiamo una finestra principale (`Tk`) e due widget essenziali: - Un `Label` che mostra il testo da digitare.
- Un `Entry` che consente l'inserimento a riga singola da parte dell'utente [3].

Il widget `Entry` fornisce metodi utili come `.get()` per leggere il contenuto corrente e `.delete()` per cancellare caratteri [4]. Inizializziamo questi widget e li posizioniamo nella finestra con i normali metodi di layout di Tkinter (ad es. `pack()`).

Gestione degli eventi di digitazione

Assegnamo al `Entry` un handler sull'evento di rilascio tasto (`<KeyRelease>`). Ogni volta che l'utente preme un tasto, nella funzione di callback controlliamo l'ultimo carattere digitato confrontandolo con il carattere corrispondente nel testo target. Se il carattere **è corretto**, lo accettiamo e avanziamo l'indice interno; se è il primo corretto, avviamo il timer con `time.perf_counter()`. Se il carattere **non è corretto**, incrementiamo il conteggio degli errori e blocchiamo l'input di quel carattere (non lo lasciamo apparire nell'Entry). In Tkinter è possibile interrompere la propagazione dell'evento di tasto restituendo la stringa `"break"` [5], in modo che il carattere sbagliato non venga inserito. Questo implementa la regola "non si può proseguire se sbaglia": l'utente deve riprovare finché non digita il carattere giusto.

Cronometro e penalità

Utilizziamo il modulo `time` per misurare il tempo trascorso. In particolare, la funzione `time.perf_counter()` restituisce un valore in secondi ad alta risoluzione [6], ideale per misurare durate brevi. Salviamo il tempo iniziale quando viene digitato il primo carattere corretto, e il tempo finale quando l'utente completa l'ultimo carattere. Il tempo trascorso reale è la differenza fra fine e inizio; aggiungiamo poi **0,5 secondi di penalità per ogni errore** conteggiato.

Codice Python d'esempio

Di seguito un esempio di codice Python strutturato in una classe per implementare questa logica con Tkinter. L'interfaccia è semplice ma funzionale: legge `testo.txt`, mostra il testo e gestisce il campo di input con validazione immediata. Al termine mostra un messaggio con il tempo totale (incluso penalità).

```
import tkinter as tk
from tkinter import messagebox
import time

class TypingApp:
    def __init__(self, root):
        root.title("Esercizio di Dattilografia")
        # Lettura del testo da file
        try:
            with open("testo.txt", "r", encoding="utf-8") as f:
                self.text = f.read().strip()
        except FileNotFoundError:
            self.text = "Errore: file testo.txt non trovato."

        # Variabili di stato
        self.index = 0        # indice del prossimo carattere da digitare
        self.errors = 0       # contatore di errori
        self.start_time = None

        # Label per il testo da digitare
        self.label = tk.Label(root, text=self.text, font=('Consolas', 14))
        self.label.pack(pady=10)

        # Entry per l'input dell'utente
        self.entry = tk.Entry(root, font=('Consolas', 14))
        self.entry.pack(pady=10)
        self.entry.focus_set()

        # Bind degli eventi: blocco del backspace e verifica carattere
        self.entry.bind("<KeyPress>", self.handle_keypress)
        self.entry.bind("<KeyRelease>", self.check_char)

    def handle_keypress(self, event):
        # Blocca l'uso del tasto BackSpace per evitare cancellazioni
        if event.keysym == "BackSpace":
```

```
            return "break"

    def check_char(self, event):
        current = self.entry.get()
        if not current:
            return
        char = current[-1]
        # Se è il primo carattere giusto, avvia il timer
        if self.index == 0 and char == self.text[0]:
            self.start_time = time.perf_counter()
        # Controllo del carattere digitato
        if self.index < len(self.text) and char == self.text[self.index]:
            self.index += 1
            # Se completato, calcola e mostra risultato
            if self.index == len(self.text):
                end_time = time.perf_counter()
                elapsed = end_time - (self.start_time or 0)
                penalty = self.errors * 0.5
                total = elapsed + penalty
                message = (f"Tempo impiegato: {elapsed:.2f} sec\n"
                           f"Errori: {self.errors}\n"
                           f"Penalità: {penalty:.2f} sec\n"
                           f"Tempo totale: {total:.2f} sec")
                messagebox.showinfo("Risultato", message)
                self.root.quit()
        else:
            # Carattere sbagliato: aggiunge penalità ed elimina il carattere
            self.errors += 1
            self.entry.delete(len(current)-1, tk.END)

if __name__ == "__main__":
    root = tk.Tk()
    app = TypingApp(root)
    root.mainloop()
```

File di esempio (`testo.txt`)

Di seguito il contenuto di un semplice file di esempio. Si possono usare testi diversi per esercitarsi.

```
La volpe marrone veloce salta sopra il cane pigro.
```

Fonti: la gestione del file in Python [1], l'uso del widget `Entry` di Tkinter [3] [4], e il metodo di misurazione del tempo con `time.perf_counter()` [6]. Inoltre, la tecnica di bloccare l'input con `return "break"` è spiegata nella documentazione di Tkinter [5].

---

[1] Python File Open
https://www.w3schools.com/python/python_file_open.asp

2 3 4 Python Tkinter - Entry Widget - GeeksforGeeks
https://www.geeksforgeeks.org/python/python-tkinter-entry-widget/

5 Events and Bindings
https://dafarry.github.io/tkinterbook/tkinter-events-and-bindings.htm

6 time — Time access and conversions — Python 3.13.5 documentation
https://docs.python.org/3/library/time.html