40 Progetti Open Source in Python

Un omaggio a tutti gli studenti di Informatica e appasionati di questa meravigliosa materia i progetti sono ben commentati in Italiano e Pubblici su Github.

Di

Luca Bocaletto

www.elektronoide.it

Indice

22. **Password Manager**
23. **Tris 2 Player**
24. **Timed Shutdown PC**
25. **Temporary Files Delete**
26. **Send Remote Commands**
27. **Orologio Mondiale con Allarme**
28. **Sincronia con Orologio Atomico**
29. **Paint Disegna Facile**
30. **Cerca File**
31. **Cerca su Wikipedia**
32. **Generatore Multiplo di Onde**
33. **Formatta Chiavette USB**
34. **Calcolatrice**
35. **Formatta SD-Card**
36. **Calendario a Eventi**
37. **Galleria Immagini**
38. **Mappe Mentali**
39. **Processi Autoavvio Windows-Autorun-Process-Manager**
40. **Synthsizer LB-1**

Gestore Appuntamenti

Il **Gestore Appuntamenti** è un software progettato per semplificare la gestione e l'organizzazione degli appuntamenti personali o professionali. Sviluppato da Luca Bocaletto, offre un'interfaccia utente intuitiva basata sulla libreria PyQt6 per la creazione di applicazioni desktop in Python.

Caratteristiche principali

- **Registrazione degli Appuntamenti:** Gli utenti possono inserire dettagli essenziali come il nome dell'appuntamento, la data e una descrizione opzionale. I dati vengono memorizzati in un database SQLite per un accesso rapido e affidabile.

- **Visualizzazione degli Appuntamenti:** Il software offre una comoda tabella che visualizza tutti gli appuntamenti registrati, inclusi ID univoci, nomi, date e descrizioni. Gli appuntamenti sono ordinati per data per una migliore organizzazione.

- **Eliminazione Facile:** Gli utenti possono eliminare gli appuntamenti selezionati direttamente dalla tabella con un semplice clic sul pulsante "Elimina".

- **Informazioni sull'Applicazione:** L'applicazione include una finestra "About" che fornisce informazioni sul creatore del software e la versione corrente.

- **Interfaccia Utente Personalizzata:** L'interfaccia utente offre un titolo colorato e una formattazione chiara per una facile lettura e navigazione.

- **Validazione dei Dati:** Il software garantisce che i campi obbligatori (nome e data) siano compilati prima di inserire un appuntamento, prevenendo così errori dati.

Il **Gestore Appuntamenti** è uno strumento ideale per chiunque abbia bisogno di tenere traccia degli impegni e degli appuntamenti quotidiani. La sua interfaccia semplice e intuitiva lo rende accessibile anche per gli utenti meno esperti, mentre la funzionalità di archiviazione dei dati in un database SQLite garantisce l'affidabilità e la persistenza delle informazioni.

Requisiti di Sistema

- Python 3.x
- PyQt6
- SQLite

Source Code: Gestione Appuntamenti

```
# Name: Gestione Appuntamenti
# Author: Bocaletto Luca Aka Elektronoide
# Importazioni dei moduli necessari
import sys
from PyQt6.QtWidgets import QApplication, QMainWindow, QVBoxLayout, QWidget, QLabel, QLineEdit, QPushButton, QTableWidget, QTableWidgetItem, QHBoxLayout, QDialog, QMessageBox, QCalendarWidget
from PyQt6.QtCore import Qt
from PyQt6.QtGui import QFont, QPalette, QColor

import sqlite3
```

```
# Definizione di una finestra di dialogo "About"
class AboutDialog(QDialog):
    def __init__(self):
        super().__init__()

        self.setWindowTitle("About")  # Imposta il titolo della finestra
        self.setGeometry(100, 100, 400, 200)  # Imposta le dimensioni della finestra

        layout = QVBoxLayout()  # Crea un layout verticale per la finestra

        about_label = QLabel("Gestore Appuntamenti v0.5 - By Bocaletto Luca")  # Etichetta con le informazioni sull'app
        layout.addWidget(about_label)  # Aggiungi l'etichetta al layout

        ok_button = QPushButton("OK")  # Crea un pulsante "OK"
        ok_button.clicked.connect(self.accept)  # Collega la pressione del pulsante alla chiusura della finestra

        layout.addWidget(ok_button)  # Aggiungi il pulsante al layout

        self.setLayout(layout)  # Imposta il layout per la finestra

# Definizione della classe principale dell'app
class AppuntamentiApp(QMainWindow):
    def __init__(self):
        super().__init__()

        self.setWindowTitle("Gestione Appuntamenti")  # Imposta il titolo della finestra principale
        self.setGeometry(100, 100, 800, 600)  # Imposta le dimensioni della finestra

        central_widget = QWidget()
        self.setCentralWidget(central_widget)

        layout = QVBoxLayout()
        central_widget.setLayout(layout)

        # Aggiungi un titolo colorato
        title_label = QLabel("Gestione Appuntamenti")  # Titolo dell'app
        title_label.setFont(QFont("Arial", 24))  # Imposta il carattere e la dimensione del titolo
        title_label.setAlignment(Qt.AlignmentFlag.AlignCenter)  # Centra il titolo
        palette = QPalette()
        palette.setColor(QPalette.ColorRole.WindowText, QColor(0, 102, 204))  # Imposta il colore del testo
        title_label.setPalette(palette)
        layout.addWidget(title_label)

        # Creazione di elementi del modulo di input
        self.nome_label = QLabel("Nome:")
        self.nome_entry = QLineEdit()
        self.data_label = QLabel("Data:")
        self.data_calendar = QCalendarWidget()
        self.descrizione_label = QLabel("Descrizione:")
```

```
    self.descrizione_entry = QLineEdit()

    form_layout = QVBoxLayout()
    form_layout.addWidget(self.nome_label)
    form_layout.addWidget(self.nome_entry)
    form_layout.addWidget(self.data_label)
    form_layout.addWidget(self.data_calendar)
    form_layout.addWidget(self.descrizione_label)
    form_layout.addWidget(self.descrizione_entry)

    button_layout = QHBoxLayout()
    self.inserisci_button = QPushButton("Inserisci")
    self.elimina_button = QPushButton("Elimina")
    button_layout.addWidget(self.inserisci_button)
    button_layout.addWidget(self.elimina_button)

    layout.addLayout(form_layout)
    layout.addLayout(button_layout)

    # Collega i pulsanti alle rispettive funzioni
    self.inserisci_button.clicked.connect(self.inserisci_appuntamento)
    self.elimina_button.clicked.connect(self.elimina_appuntamento)

    # Creazione di una tabella per visualizzare gli appuntamenti
    self.table = QTableWidget()
    self.table.setColumnCount(4)
    self.table.setHorizontalHeaderLabels(["ID", "Nome", "Data", "Descrizione"])
    layout.addWidget(self.table)

    # Connessione al database SQLite
    self.conn = sqlite3.connect("appuntamenti.db")
    self.cursor = self.conn.cursor()

    # Creazione della tabella se non esiste
    self.crea_tabella_appuntamenti()

    # Elenco degli appuntamenti nella tabella
    self.elenca_appuntamenti()

    # Aggiungi il pulsante "About" alla barra dei menu
    about_action = self.menuBar().addAction("About")
    about_action.triggered.connect(self.mostra_about)

def crea_tabella_appuntamenti(self):
    # Crea la tabella nel database se non esiste
    self.cursor.execute('''
        CREATE TABLE IF NOT EXISTS appuntamenti (
            id INTEGER PRIMARY KEY,
            nome TEXT NOT NULL,
            data DATE NOT NULL,
            descrizione TEXT
        )
```

```
        ''')
        self.conn.commit()

    def inserisci_appuntamento(self):
        # Ottieni i dati dall'input e inseriscili nel database
        nome = self.nome_entry.text()
        data = self.data_calendar.selectedDate().toString("yyyy-MM-dd")
        descrizione = self.descrizione_entry.text()
        if nome and data:
            self.cursor.execute("INSERT INTO appuntamenti (nome, data, descrizione) VALUES
(?, ?, ?)",
                                (nome, data, descrizione))
            self.conn.commit()
            self.elenca_appuntamenti()  # Aggiorna la tabella degli appuntamenti
            self.nome_entry.clear()  # Cancella l'input del nome
            self.descrizione_entry.clear()  # Cancella l'input della descrizione
        else:
            QMessageBox.warning(self, "Attenzione", "Nome e Data sono campi obbligatori.")

    def elimina_appuntamento(self):
        # Elimina un appuntamento dalla tabella
        riga_selezionata = self.table.currentRow()
        if riga_selezionata >= 0:
            id_selezionato = self.table.item(riga_selezionata, 0).text()
            self.cursor.execute("DELETE FROM appuntamenti WHERE id=?", (id_selezionato,))
            self.conn.commit()
            self.elenca_appuntamenti()  # Aggiorna la tabella degli appuntamenti

    def elenca_appuntamenti(self):
        # Ottieni gli appuntamenti dal database e visualizzali nella tabella
        self.cursor.execute("SELECT * FROM appuntamenti")
        appuntamenti = self.cursor.fetchall()
        self.table.setRowCount(0)  # Cancella tutte le righe attuali dalla tabella
        for i, appuntamento in enumerate(appuntamenti):
            self.table.insertRow(i)  # Inserisci una nuova riga nella tabella
            for j, col in enumerate(appuntamento):
                item = QTableWidgetItem(str(col))
                item.setFlags(item.flags() ^ Qt.ItemFlag.ItemIsEditable)  # Impedisce la modifica diretta
dei dati
                self.table.setItem(i, j, item)

    def mostra_about(self):
        about_dialog = AboutDialog()
        about_dialog.exec()

def main():
    app = QApplication(sys.argv)
    window = AppuntamentiApp()
    window.show()
    sys.exit(app.exec())

if __name__ == "__main__":
```

```
main()
```

IP Port Scanner

Descrizione

Il software "IP Port Scanner" è un'applicazione Python basata su Tkinter che consente agli utenti di eseguire la scansione degli indirizzi IP e delle porte in una rete locale. Questa applicazione fornisce un'interfaccia utente intuitiva per eseguire scansioni IP e verificare lo stato delle porte aperte o chiuse su un determinato host.

Caratteristiche principali

- **Scansione IP:** Gli utenti possono specificare un intervallo di indirizzi IP e avviare una

scansione per ottenere informazioni sugli host presenti nella rete.

- **Scansione delle porte:** Dopo aver selezionato un host dalla lista risultante, è possibile eseguire una scansione delle porte per determinare quali porte sono aperte e chiuse su quell'host.

- **Interfaccia utente semplice:** L'applicazione presenta un'interfaccia utente intuitiva con pulsanti per avviare, interrompere e visualizzare i risultati delle scansioni.

- **Risultati dettagliati:** I risultati delle scansioni vengono visualizzati in un albero con informazioni dettagliate sugli indirizzi IP, gli hostname e lo stato delle porte.

Utilizzo

1. Specifica un intervallo di indirizzi IP (ad esempio, 192.168.1.1-192.168.1.254) nell'apposita casella di input.

2. Avvia la scansione IP per individuare gli host nella rete.

3. Seleziona un host dalla lista risultante.

4. Specifica un intervallo di porte (ad esempio, 1-1024) nell'apposita casella di input.

5. Avvia la scansione delle porte per verificare lo stato delle porte su quell'host.

6. I risultati della scansione delle porte vengono visualizzati nell'albero.

Source Code: IP Port Scanner

```
# Software Name: IP Port Scanner
# Author: Bocaletto Luca
# Web Site: https://www.elektronoide.it
import tkinter as tk
import socket
import threading
import tkinter.ttk as ttk
from tkinter import messagebox

# Classe principale dell'applicazione di scansione IP
class IPScannerApp:
    def __init__(self, root):
        self.root = root
```

```
        self.root.title("IP Port Scanner")

        # Aggiungi un label con il titolo del software
        software_label = tk.Label(root, text="IP Port Scanner", font=("Helvetica", 16))
        software_label.grid(row=0, column=0, columnspan=3, padx=10, pady=10)

        # Variabili per il controllo dello stato della scansione
        self.scan_in_progress = False
        self.scan_thread = None

        # Etichette e casella di input per l'intervallo IP
        self.ip_range_label = tk.Label(root, text="IP Range (e.g., 192.168.1.1-192.168.1.254):")
        self.ip_range_label.grid(row=1, column=0, columnspan=3, padx=10, pady=10)

        self.ip_range_entry = tk.Entry(root)
        self.ip_range_entry.grid(row=2, column=0, columnspan=3, padx=10, pady=10)

        # Pulsanti
        self.scan_button = tk.Button(root, text="Scan IPs", command=self.start_ip_scan)
        self.scan_button.grid(row=3, column=0, padx=10, pady=10)

        self.stop_button = tk.Button(root, text="Stop", command=self.stop_scan,
state=tk.DISABLED)
        self.stop_button.grid(row=3, column=1, padx=10, pady=10)

        self.port_scan_button = tk.Button(root, text="Port Scan", command=self.start_port_scan,
state=tk.DISABLED)
        self.port_scan_button.grid(row=3, column=2, padx=10, pady=10)

        # Albero per visualizzare i risultati
        self.tree = ttk.Treeview(root, columns=("IP", "Hostname"), show="headings", height=15)
        self.tree.heading("IP", text="IP")
        self.tree.heading("Hostname", text="Hostname")
        self.tree.grid(row=4, column=0, columnspan=3, padx=10, pady=10)

        # Lista degli IP selezionati
        self.selected_ips = []

    # Funzione per avviare la scansione IP
    def start_ip_scan(self):
        if self.scan_in_progress:
            messagebox.showinfo("Info", "Scan already in progress")
            return

        self.scan_in_progress = True
        self.scan_button.config(state=tk.DISABLED)
        self.stop_button.config(state=tk.NORMAL)

        # Ottieni l'intervallo IP dall'input
        ip_range = self.ip_range_entry.get()
        start_ip, end_ip = ip_range.split('-')
```

```
    try:
        start_ip_int = int(start_ip.split('.')[-1])
        end_ip_int = int(end_ip.split('.')[-1])

        # Cancella i risultati precedenti nell'albero
        self.tree.delete(*self.tree.get_children())

        # Funzione per eseguire la scansione di un singolo IP
        def scan_ip(ip):
            try:
                hostname = socket.gethostbyaddr(ip)
            except socket.herror:
                hostname = ("N/A",)

            # Inserisci i risultati nell'albero
            self.tree.insert("", "end", values=(ip, hostname[0]))

        # Funzione per eseguire la scansione nell'intervallo specificato
        def scan_range():
            for ip_int in range(start_ip_int, end_ip_int + 1):
                if not self.scan_in_progress:
                    break
                current_ip = f"192.168.1.{ip_int}"  # Modifica con la tua rete
                scan_ip(current_ip)

            # Alla fine della scansione, ripristina i pulsanti
            self.scan_in_progress = False
            self.scan_button.config(state=tk.NORMAL)
            self.stop_button.config(state=tk.DISABLED)
            self.port_scan_button.config(state=tk.NORMAL)

        # Avvia la scansione in un thread separato
        self.scan_thread = threading.Thread(target=scan_range)
        self.scan_thread.start()

    except ValueError:
        messagebox.showerror("Error", "Invalid IP range")

# Funzione per interrompere la scansione IP
def stop_scan(self):
    self.scan_in_progress = False
    if self.scan_thread and self.scan_thread.is_alive():
        self.scan_thread.join()

    # Ripristina i pulsanti
    self.scan_button.config(state=tk.NORMAL)
    self.stop_button.config(state=tk.DISABLED)
    self.port_scan_button.config(state=tk.DISABLED)

# Funzione per avviare la scansione delle porte
def start_port_scan(self):
    selected_item = self.tree.selection()
```

```
        if not selected_item:
            messagebox.showinfo("Info", "Select IPs to scan ports")
            return

        # Ottieni l'IP selezionato dall'albero
        selected_ip = self.tree.item(selected_item[0], "values")[0]

        # Crea una finestra separata per la scansione delle porte
        port_scan_window = tk.Toplevel(self.root)
        PortScanner(port_scan_window, selected_ip)

# Classe per la scansione delle porte
class PortScanner:
    def __init__(self, root, ip):
        self.root = root
        self.root.title(f"Port Scanner - IP: {ip}")

        self.ip = ip

        # Etichette e casella di input per l'intervallo delle porte
        self.port_range_label = tk.Label(root, text="Port Range (e.g., 1-1024):")
        self.port_range_label.grid(row=0, column=0, columnspan=2, padx=10, pady=10)

        self.port_range_entry = tk.Entry(root)
        self.port_range_entry.grid(row=1, column=0, columnspan=2, padx=10, pady=10)

        # Pulsanti
        self.scan_button = tk.Button(root, text="Scan Ports", command=self.start_port_scan)
        self.scan_button.grid(row=2, column=0, padx=10, pady=10)

        self.stop_button = tk.Button(root, text="Stop", command=self.stop_port_scan,
state=tk.DISABLED)
        self.stop_button.grid(row=2, column=1, padx=10, pady=10)

        # Albero per visualizzare i risultati della scansione delle porte
        self.tree = ttk.Treeview(root, columns=("Port", "Status"), show="headings", height=15)
        self.tree.heading("Port", text="Port")
        self.tree.heading("Status", text="Status")
        self.tree.grid(row=3, column=0, columnspan=2, padx=10, pady=10)

        # Variabili per il controllo dello stato della scansione delle porte
        self.scan_in_progress = False
        self.scan_thread = None

    # Funzione per avviare la scansione delle porte
    def start_port_scan(self):
        if self.scan_in_progress:
            messagebox.showinfo("Info", "Scan already in progress")
            return

        self.scan_in_progress = True
        self.scan_button.config(state=tk.DISABLED)
```

```
        self.stop_button.config(state=tk.NORMAL)

        # Ottieni l'intervallo delle porte dall'input
        port_range = self.port_range_entry.get()
        start_port, end_port = map(int, port_range.split('-'))

        # Cancella i risultati precedenti nell'albero
        self.tree.delete(*self.tree.get_children())

        # Funzione per eseguire la scansione di una singola porta
        def scan_port(port):
            sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            sock.settimeout(1)
            result = sock.connect_ex((self.ip, port))
            sock.close()
            if result == 0:
                status = "Open"
            else:
                status = "Closed"
            self.tree.insert("", "end", values=(port, status))
            self.tree.see(self.tree.get_children()[-1])

        # Funzione per eseguire la scansione delle porte nell'intervallo specificato
        def scan_ports_range():
            for port in range(start_port, end_port + 1):
                if not self.scan_in_progress:
                    break
                scan_port(port)

            # Alla fine della scansione, ripristina i pulsanti
            self.scan_in_progress = False
            self.scan_button.config(state=tk.NORMAL)
            self.stop_button.config(state=tk.DISABLED)

        # Avvia la scansione delle porte in un thread separato
        self.scan_thread = threading.Thread(target=scan_ports_range)
        self.scan_thread.start()

    # Funzione per interrompere la scansione delle porte
    def stop_port_scan(self):
        self.scan_in_progress = False
        if self.scan_thread and self.scan_thread.is_alive():
            self.scan_thread.join()

        # Ripristina i pulsanti
        self.scan_button.config(state=tk.NORMAL)
        self.stop_button.config(state=tk.DISABLED)

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

Rubrica Telefonica

Descrizione

Questa è un'applicazione Python che implementa una semplice rubrica telefonica con un'interfaccia grafica utente (GUI) utilizzando PyQt5. L'applicazione consente agli utenti di aggiungere, visualizzare ed eliminare contatti telefonici. Inoltre, offre una funzione "About" per visualizzare informazioni sull'autore dell'applicazione.

Funzionalità

- Aggiunta di nuovi contatti con nome e numero telefonico.
- Visualizzazione dei contatti nella tabella.

- Eliminazione di contatti selezionati.
- Visualizzazione delle informazioni sull'autore dell'applicazione.
- Salvataggio dei contatti su un file per il caricamento successivo all'avvio dell'applicazione.

Requisiti

- Python 3.x
- PyQt5 (installabile tramite pip: `pip install PyQt5`)

Utilizzo

1. Assicurarsi che Python sia installato nel sistema.
2. Installare PyQt5 se non è già presente: `pip install PyQt5`.
3. Eseguire il file `rubrica_telefonica.py` con Python.

L'applicazione si aprirà e verrà visualizzata l'interfaccia utente della rubrica telefonica.

Source Code: Rubrica Telefonica

```
# Software Name: PhoneBook
# Autrhor: Bocaletto Luca
# Web Site: https://www.elektronoide.it
import sys
from PyQt5.QtWidgets import QApplication, QWidget, QVBoxLayout,
QPushButton, QTableWidget, QTableWidgetItem, QLineEdit, QMessageBox,
QHeaderView, QLabel, QDialog
from PyQt5 import QtCore

class Contatto:
    def __init__(self, nome, telefono):
        self.nome = nome
        self.telefono = telefono
        self.selezionato = False

class AboutDialog(QDialog):
    def __init__(self):
        super().__init__()

        self.setWindowTitle('Informazioni')
        self.setGeometry(200, 200, 300, 150)

        layout = QVBoxLayout()
```

```
        informazioni_label = QLabel('Informazioni')
        autore_label = QLabel('Autore: Bocaletto Luca')
        email_label = QLabel('Email: your@email.com')
        website_label = QLabel('https://www.elektronoide.it')

        layout.addWidget(informazioni_label)
        layout.addWidget(autore_label)
        layout.addWidget(email_label)

        self.setLayout(layout)

class ContattiApp(QWidget):
    def __init__(self):
        super().__init__()
        self.initUI()

    def initUI(self):
        # Impostazione della finestra principale
        self.setWindowTitle('Rubrica Telefonica')
        self.setGeometry(100, 100, 600, 300)

        # Creazione del layout principale
        self.layout = QVBoxLayout()

        # Aggiunta di un titolo nella GUI
        self.titolo_label = QLabel('Rubrica Telefonica')
        self.titolo_label.setAlignment(QtCore.Qt.AlignmentFlag.AlignCenter)  # Centro
il testo
        self.titolo_label.setStyleSheet("font-size: 24px; color: blue;")  # Imposta colore
e dimensione del testo

        # Creazione dei campi di inserimento per Nome e Telefono
        self.nome_entry = QLineEdit()
        self.telefono_entry = QLineEdit()

        # Creazione dei pulsanti per Aggiungi Contatto, Elimina Contatti e About
        self.aggiungi_button = QPushButton('Aggiungi Contatto')
        self.elimina_button = QPushButton('Elimina Contatti')
        self.about_button = QPushButton('About')

        # Creazione della tabella per visualizzare i contatti
        self.tabella_contatti = QTableWidget()
        self.tabella_contatti.setColumnCount(3)  # Tre colonne: Nome, Telefono,
Selezione
        self.tabella_contatti.setHorizontalHeaderLabels(['Nome', 'Telefono', 'Selezione'])
```

```
self.tabella_contatti.horizontalHeader().setSectionResizeMode(QHeaderView.Stretch
)  # Ridimensionamento delle colonne

        # Aggiunta dei widget al layout principale
        self.layout.addWidget(self.titolo_label)
        self.layout.addWidget(self.nome_entry)
        self.layout.addWidget(self.telefono_entry)
        self.layout.addWidget(self.aggiungi_button)
        self.layout.addWidget(self.elimina_button)
        self.layout.addWidget(self.about_button)
        self.layout.addWidget(self.tabella_contatti)

        # Collegamento dei pulsanti alle funzioni
        self.aggiungi_button.clicked.connect(self.aggiungi_contatto)
        self.elimina_button.clicked.connect(self.elimina_contatti)
        self.about_button.clicked.connect(self.mostra_about)

        # Impostazione del layout principale
        self.setLayout(self.layout)

        # Caricamento dei contatti all'avvio dell'applicazione
        self.carica_contatti()

    def mostra_about(self):
        dialog = AboutDialog()
        dialog.exec_()

    def aggiungi_contatto(self):
        # Funzione per aggiungere un nuovo contatto
        nome = self.nome_entry.text().strip()  # Rimuove gli spazi vuoti iniziali e finali
        telefono = self.telefono_entry.text().strip()

        if not nome or not telefono:
            # Se uno o entrambi i campi sono vuoti, mostra un messaggio di avviso
            QMessageBox.critical(self, 'Errore', 'Nome e Telefono devono essere
riempiti.')
            return  # Esce dalla funzione senza aggiungere il contatto

        # Aggiunge il contatto alla lista
        contatto = Contatto(nome, telefono)
        self.contatti.append(contatto)

        # Ordina la lista in base ai nomi
        self.contatti.sort(key=lambda x: x.nome)
```

```
        # Aggiorna la tabella
        self.aggiorna_tabella()

        self.nome_entry.clear()
        self.telefono_entry.clear()
        self.salva_contatti()

    def carica_contatti(self):
        # Funzione per caricare i contatti da un file all'avvio
        self.contatti = []  # Lista per mantenere i contatti

        try:
            with open("contatti.txt", "r") as file:
                for linea in file:
                    parts = linea.strip().split(': ')
                    if len(parts) == 2:
                        nome, telefono = parts
                        contatto = Contatto(nome, telefono)
                        self.contatti.append(contatto)

            # Ordina la lista in base ai nomi
            self.contatti.sort(key=lambda x: x.nome)

            # Aggiorna la tabella
            self.aggiorna_tabella()
        except FileNotFoundError:
            pass

    def salva_contatti(self):
        # Funzione per salvare i contatti su un file
        try:
            with open("contatti.txt", "w") as file:
                for contatto in self.contatti:
                    file.write(f"{contatto.nome}: {contatto.telefono}\n")
        except IOError:
            QMessageBox.critical(self, 'Errore', 'Impossibile salvare i contatti su file.')

    def aggiorna_tabella(self):
        # Funzione per aggiornare la tabella con i contatti attuali
        self.tabella_contatti.setRowCount(len(self.contatti))

        for row, contatto in enumerate(self.contatti):
            self.tabella_contatti.setItem(row, 0, QTableWidgetItem(contatto.nome))
            self.tabella_contatti.setItem(row, 1, QTableWidgetItem(contatto.telefono))
```

```
            # Aggiunge una casella di controllo alla terza colonna
            checkbox = QTableWidgetItem()
            checkbox.setFlags(checkbox.flags() | QtCore.Qt.ItemIsUserCheckable)
            checkbox.setCheckState(QtCore.Qt.Checked if contatto.selezionato else
QtCore.Qt.Unchecked)
            self.tabella_contatti.setItem(row, 2, checkbox)

    def elimina_contatti(self):
        # Funzione per eliminare i contatti selezionati
        contatti_da_eliminare = []

        for row, contatto in enumerate(self.contatti):
            checkbox = self.tabella_contatti.item(row, 2)
            if checkbox and checkbox.checkState() == QtCore.Qt.Checked:
                contatti_da_eliminare.append(contatto)

        for contatto in contatti_da_eliminare:
            self.contatti.remove(contatto)

        # Aggiorna la tabella
        self.aggiorna_tabella()

        # Salva i contatti aggiornati
        self.salva_contatti()

if __name__ == '__main__':
    app = QApplication(sys.argv)
    ex = ContattiApp()
    ex.show()
    sys.exit(app.exec_())
```

Ham Radio Logger

Autore: Bocaletto Luca
Sito Web: https://www.elektronoide.it

Descrizione

Ham Radio Logger è un software specializzato progettato da Bocaletto Luca per gli appassionati di radioamatore, finalizzato a semplificare la registrazione e la gestione delle comunicazioni radio in modo efficiente ed efficace. Questa applicazione si rivolge a un pubblico di radioamatori, operatori di stazioni radioamatoriali e appassionati di radiofonia, offrendo una serie di strumenti potenti per la catalogazione e la consultazione dei contatti radio effettuati.

Caratteristiche principali

- **Registrazione dei Contatti:** Ham Radio Logger consente agli utenti di registrare in modo semplice e dettagliato ogni contatto radio effettuato. È possibile inserire informazioni come la data, l'orario, la frequenza, la modalità di comunicazione e i dettagli del contatto radio.

- **Gestione delle Stazioni:** Gli utenti possono mantenere un elenco dettagliato delle stazioni radio con cui hanno comunicato. Questo include informazioni quali il nominativo, la posizione geografica, le bande di frequenza preferite e altre note pertinenti.

- **Ricerca e Filtri:** Il software offre potenti strumenti di ricerca e filtri che consentono agli utenti di accedere rapidamente a informazioni specifiche sui contatti radio precedenti. È possibile effettuare ricerche per data, banda di frequenza, nominativo e altre variabili.

- **Statistiche e Report:** Ham Radio Logger genera automaticamente statistiche dettagliate sui contatti radio registrati. Gli utenti possono visualizzare report personalizzati che mostrano le proprie prestazioni, le stazioni più frequenti o altre metriche di interesse.

- **Integrazione con Mappe:** L'applicazione offre la possibilità di integrare le informazioni geografiche, consentendo agli utenti di visualizzare la posizione delle stazioni con cui hanno comunicato su mappe interattive.

- **Backup e Ripristino:** Per garantire la sicurezza dei dati, il software permette di eseguire facilmente il backup delle informazioni registrate e il ripristino in caso di perdita di dati.

- **Interfaccia Utente Intuitiva:** Ham Radio Logger è progettato con un'interfaccia utente intuitiva che consente agli utenti di accedere facilmente a tutte le funzionalità del software senza complicazioni.

Source Code: Ham-Radio Logger

```
# Nome Software: Ham Radio Logger
# Autore: Bocaletto Luca
# Sito Web: https://www.elektronoide.it
import sys
import sqlite3
from PyQt5.QtWidgets import QApplication, QMainWindow, QWidget,
QVBoxLayout, QPushButton, QTableWidget, QTableWidgetItem, QLineEdit,
QLabel
from PyQt5.QtCore import Qt
from PyQt5.QtGui import QFont
```

```
class HamRadioLogger(QMainWindow):
    def __init__(self):
        super().__init__()
        self.initUI()

    def initUI(self):
        # Imposta il titolo della finestra
        self.setWindowTitle('Logger Radioamatoriale')
        # Imposta le dimensioni della finestra
        self.setGeometry(100, 100, 800, 600)

        # Crea un widget centrale
        self.central_widget = QWidget()
        self.setCentralWidget(self.central_widget)

        # Crea un layout verticale per organizzare i widget
        self.layout = QVBoxLayout()

        # Add the title label and set its font size
        self.title_label = QLabel('Ham Radio Logger')
        font = QFont()
        font.setPointSize(20)  # Set font size to 20
        self.title_label.setFont(font)
        self.title_label.setAlignment(Qt.AlignCenter)
        self.layout.addWidget(self.title_label)

        # Crea etichette e campi di input per i dati
        self.call_label = QLabel('Indicativo di chiamata:')
        self.call_input = QLineEdit()
        self.call_input.setPlaceholderText('Es. IZ4XYZ')
        self.freq_label = QLabel('Frequenza:')
        self.freq_input = QLineEdit()
        self.freq_input.setPlaceholderText('Es. 144.800 MHz')
        self.date_label = QLabel('Data:')
        self.date_input = QLineEdit()
        self.date_input.setPlaceholderText('Es. 2023-01-15')
        self.time_label = QLabel('Ora:')
        self.time_input = QLineEdit()
        self.time_input.setPlaceholderText('Es. 14:30')
        self.power_label = QLabel('Potenza del segnale:')
        self.power_input = QLineEdit()
        self.power_input.setPlaceholderText('Es. 50 Watts')

        # Crea bottoni
```

```
        self.add_button = QPushButton('Aggiungi')
        self.update_button = QPushButton('Modifica')
        self.delete_button = QPushButton('Elimina')
        self.export_button = QPushButton('Esporta in CSV')

        # Crea una tabella per visualizzare i dati
        self.table = QTableWidget()
        self.table.setColumnCount(6)
        self.table.setHorizontalHeaderLabels(['ID', 'Indicativo di chiamata', 'Frequenza',
'Data', 'Ora', 'Potenza del segnale'])
        self.table.horizontalHeader().setStretchLastSection(True)

        # Aggiungi i widget al layout
        self.layout.addWidget(self.call_label)
        self.layout.addWidget(self.call_input)
        self.layout.addWidget(self.freq_label)
        self.layout.addWidget(self.freq_input)
        self.layout.addWidget(self.date_label)
        self.layout.addWidget(self.date_input)
        self.layout.addWidget(self.time_label)
        self.layout.addWidget(self.time_input)
        self.layout.addWidget(self.power_label)
        self.layout.addWidget(self.power_input)
        self.layout.addWidget(self.add_button)
        self.layout.addWidget(self.update_button)
        self.layout.addWidget(self.delete_button)
        self.layout.addWidget(self.export_button)
        self.layout.addWidget(self.table)

        # Imposta il layout del widget centrale
        self.central_widget.setLayout(self.layout)

        # Collega i bottoni alle funzioni corrispondenti
        self.add_button.clicked.connect(self.add_entry)
        self.update_button.clicked.connect(self.update_entry)
        self.delete_button.clicked.connect(self.delete_entry)
        self.export_button.clicked.connect(self.export_to_csv)

        # Connetti al database e crea la tabella 'log'
        self.conn = sqlite3.connect('radio_logger.db')
        self.cursor = self.conn.cursor()
        self.cursor.execute('''CREATE TABLE IF NOT EXISTS log (
                            id INTEGER PRIMARY KEY,
                            call_sign TEXT,
                            frequency TEXT,
```

```
                    date TEXT,
                    time TEXT,
                    signal_power TEXT)"')
        self.conn.commit()

        # Carica i dati nella tabella
        self.update_entry_list()

    def update_entry_list(self):
        # Esegue una query per ottenere tutti i dati dalla tabella 'log'
        self.cursor.execute('SELECT * FROM log')
        data = self.cursor.fetchall()
        # Imposta il numero di righe nella tabella
        self.table.setRowCount(len(data))
        # Popola la tabella con i dati
        for i, row in enumerate(data):
            for j, item in enumerate(row):
                self.table.setItem(i, j, QTableWidgetItem(str(item)))

    def add_entry(self):
        # Ottiene i dati dai campi di input
        call = self.call_input.text()
        freq = self.freq_input.text()
        date = self.date_input.text()
        time = self.time_input.text()
        power = self.power_input.text()
        # Esegue una query per inserire i dati nella tabella
        self.cursor.execute('INSERT INTO log (call_sign, frequency, date, time,
signal_power) VALUES (?, ?, ?, ?, ?)', (call, freq, date, time, power))
        self.conn.commit()
        # Aggiorna la tabella e cancella i campi di input
        self.update_entry_list()
        self.clear_inputs()

    def update_entry(self):
        # Ottiene la riga selezionata nella tabella
        current_row = self.table.currentRow()
        if current_row == -1:
            return
        # Ottiene i dati dai campi di input
        call = self.call_input.text()
        freq = self.freq_input.text()
        date = self.date_input.text()
        time = self.time_input.text()
        power = self.power_input.text()
```

```
        # Ottiene l'ID dell'elemento selezionato
        entry_id = self.table.item(current_row, 0).text()
        # Esegue una query per aggiornare i dati
        self.cursor.execute('UPDATE log SET call_sign=?, frequency=?, date=?,
time=?, signal_power=? WHERE id=?', (call, freq, date, time, power, entry_id))
        self.conn.commit()
        # Aggiorna la tabella e cancella i campi di input
        self.update_entry_list()
        self.clear_inputs()

    def delete_entry(self):
        # Ottiene la riga selezionata nella tabella
        current_row = self.table.currentRow()
        if current_row == -1:
            return
        # Ottiene l'ID dell'elemento selezionato
        entry_id = self.table.item(current_row, 0).text()
        # Esegue una query per eliminare l'elemento
        self.cursor.execute('DELETE FROM log WHERE id=?', (entry_id,))
        self.conn.commit()
        # Aggiorna la tabella
        self.update_entry_list()
        # Cancella i campi di input
        self.clear_inputs()

    def export_to_csv(self):
        # Apre un file CSV e scrive i dati dal database
        with open('radio_logger.csv', 'w') as f:
            self.cursor.execute('SELECT * FROM log')
            data = self.cursor.fetchall()
            headers = [description[0] for description in self.cursor.description]
            f.write(','.join(headers) + '\n')
            for row in data:
                f.write(','.join(map(str, row)) + '\n')

    def clear_inputs(self):
        # Cancella i campi di input
        self.call_input.clear()
        self.freq_input.clear()
        self.date_input.clear()
        self.time_input.clear()
        self.power_input.clear()

def main():
    app = QApplication(sys.argv)
```

```
    ex = HamRadioLogger()
    ex.show()
    sys.exit(app.exec_())

if __name__ == '__main__':
    main()
```

MIDI Player

Descrizione

Il **MIDI Player** è un'applicazione desktop che consente agli utenti di caricare e riprodurre file MIDI. L'applicazione offre un'interfaccia utente intuitiva per selezionare un file MIDI, controllare la riproduzione, gestire le periferiche MIDI e monitorare il tempo di riproduzione corrente.

Caratteristiche Principali

- **Riproduzione di File MIDI**: Gli utenti possono aprire e riprodurre file MIDI selezionando il file desiderato tramite il pulsante "Apri MIDI File".

- **Controllo della Riproduzione**: L'applicazione offre un pulsante "Play" per avviare la riproduzione o mettere in pausa la riproduzione se il file MIDI è già in riproduzione. Il pulsante "Stop" consente di interrompere la riproduzione e reimpostare la posizione di riproduzione.

- **Selezione della Periferica MIDI**: Gli utenti possono selezionare la periferica MIDI di destinazione tramite una casella combinata ("Seleziona Periferica"). Questo consente di indirizzare l'output MIDI verso la periferica desiderata.

- **Visualizzazione del File MIDI**: L'applicazione mostra il nome del file MIDI attualmente selezionato nell'etichetta "File" e il tempo di riproduzione corrente nella parte inferiore della finestra.

- **Visualizzazione delle Note MIDI**: Durante la riproduzione, l'applicazione visualizza le note MIDI correnti nella parte centrale della finestra.

- **Controllo del Volume MIDI**: È possibile regolare il volume MIDI utilizzando una barra di scorrimento orizzontale. Questo consente di variare l'intensità del suono durante la riproduzione.

- **Gestione degli Errori**: L'applicazione è in grado di gestire gli errori durante la riproduzione e mostra eventuali messaggi di errore nella parte inferiore della finestra.

Utilizzo

1. Avvia l'applicazione "MIDI Player".
2. Seleziona una periferica MIDI di destinazione dall'elenco a discesa "Seleziona Periferica".
3. Carica un file MIDI utilizzando il pulsante "Apri MIDI File".
4. Utilizza il pulsante "Play" per avviare o mettere in pausa la riproduzione.
5. Utilizza il pulsante "Stop" per interrompere la riproduzione.
6. Regola il volume utilizzando la barra di scorrimento.
7. Durante la riproduzione, le note MIDI correnti vengono visualizzate nella finestra.

Source Code – Midi Player

```
# Name: Player Midi
# Author: Bocaletto Luca
# Web Site: https://www.elektronoide.it
import tkinter as tk
from tkinter import filedialog
from tkinter import ttk
import mido
import threading
import time
```

```
class MidiPlayer:
    def __init__(self, app):
        self.app = app
        self.playing = False
        self.paused = False
        self.port = None
        self.file_path = None
        self.available_ports = mido.get_output_names()
        self.current_time = tk.StringVar(value="0:00")  # Variabile per il tempo corrente
        self.note_display = tk.Label(app.root, text="")  # Etichetta per la visualizzazione
delle note
        self.volume = 64  # Aggiunto controllo del volume con valore predefinito

    def open_file(self):
        file_path = filedialog.askopenfilename(filetypes=[("MIDI Files", "*.mid")])
        if file_path:
            self.file_path = file_path
            self.app.update_file_label()  # Aggiorna il nome del file selezionato

    def select_port(self, selected_port):
        if self.playing:
            return
        if selected_port:
            self.port = mido.open_output(selected_port)

    def play(self):
        if self.playing:
            self.paused = not self.paused
            self.app.update_play_button_text()
        else:
            try:
                if not self.port:
                    self.port = mido.open_output()
                if not self.file_path:
                    return
                self.playing = True
                self.app.update_play_button_text()
                self.play_thread = threading.Thread(target=self.play_midi)
                self.play_thread.start()
            except OSError as e:
                self.stop()
                self.app.update_status(f"Errore durante la riproduzione MIDI: {str(e)}")

    def play_midi(self):
        try:
```

```
            with mido.MidiFile(self.file_path) as midi_file:
                start_time = time.time()
                for msg in midi_file.play():
                    if not self.playing:
                        break
                    while self.paused:
                        time.sleep(0.1)
                    if msg.type == "control_change":
                        msg_with_volume = msg.copy()
                        msg_with_volume.control = 7  # Numero di controllo del volume
MIDI
                        msg_with_volume.value = self.volume  # Imposta il valore del volume
                        self.port.send(msg_with_volume)
                    else:
                        self.port.send(msg)
                    current_time = time.time() - start_time
                    self.current_time.set(self.format_time(current_time))
                    if msg.type == "note_on":
                        self.note_display.config(text=f"Note: {msg.note}")
                    time.sleep(msg.time)
        except OSError as e:
            self.stop()
            self.app.update_status(f"Errore durante la riproduzione MIDI: {str(e)}")

    def stop(self):
        self.playing = False
        if self.port:
            self.port.reset()
        self.port = None
        self.app.update_play_button_text()

        # Reimposta i valori delle note e del tempo
        self.current_time.set("0:00")
        self.note_display.config(text="")

    def format_time(self, seconds):
        minutes, seconds = divmod(int(seconds), 60)
        return f"{minutes}:{seconds:02}"

class MidiPlayerApp:
    def __init__(self, root):
        self.root = root
        self.root.title("MIDI Player")
        self.player = MidiPlayer(self)
```

```
        self.play_button = tk.Button(root, text="Play", command=self.player.play)
        self.stop_button = tk.Button(root, text="Stop", command=self.player.stop)
        self.open_button = tk.Button(root, text="Apri MIDI File",
command=self.player.open_file)

        self.port_label = tk.Label(root, text="Seleziona Periferica")
        self.port_combobox = ttk.Combobox(root, values=self.player.available_ports,
state="readonly")
        self.port_combobox.bind("<<ComboboxSelected>>", self.select_port)

        # Aggiungi una etichetta per il nome del file selezionato
        self.file_label = tk.Label(root, text="File: None")

        # Aggiungi una etichetta per la visualizzazione del tempo corrente
        self.time_label = tk.Label(root, textvariable=self.player.current_time)

        self.status_label = tk.Label(root, text="")

        # Aggiungi un controllo del volume
        self.volume_label = tk.Label(root, text="Volume")
        self.volume_scale = ttk.Scale(root, from_=0, to=127,
variable=self.player.volume, orient="horizontal")

        self.play_button.pack()
        self.stop_button.pack()
        self.open_button.pack()
        self.port_label.pack()
        self.port_combobox.pack()
        self.file_label.pack()
        self.time_label.pack()
        self.player.note_display.pack()
        self.status_label.pack()

        # Mostra il controllo del volume
        self.volume_label.pack()
        self.volume_scale.pack()

    def select_port(self, event):
        selected_port = self.port_combobox.get()
        self.player.select_port(selected_port)

    def update_play_button_text(self):
        if self.player.playing:
            self.play_button.config(text="Pause" if not self.player.paused else "Riprendi")
        else:
```

```
            self.play_button.config(text="Play")

    def update_file_label(self):
        if self.player.file_path:
            file_name = self.player.file_path.split("/")[-1]
            self.file_label.config(text="File: " + file_name)
        else:
            self.file_label.config(text="File: None")

    def update_status(self, text):
        self.status_label.config(text=text)

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

7bl Compressore, decompressore con Crittografia

Autore: Luca Bocaletto

Descrizione

Questo software è progettato per aiutare gli utenti a comprimere e crittografare i loro file, rendendoli più sicuri e riducendo lo spazio di archiviazione necessario. Ecco le principali funzionalità offerte:

- **Comprimere file:** Gli utenti possono selezionare i file che desiderano comprimere. Il software utilizza la libreria di compressione zlib per ridurre le dimensioni dei file,

utilizzando il miglior livello di compressione disponibile.

- **Crittografare file:** Il software consente agli utenti di specificare una chiave di crittografia. Utilizza quindi la libreria di crittografia Fernet per crittografare i dati compressi. La crittografia offre un ulteriore livello di sicurezza ai file, proteggendoli da accessi non autorizzati.

- **Estrarre file:** Gli utenti possono anche utilizzare il software per estrarre file da un archivio crittografato. Questa operazione coinvolge la decrittografia dei dati e la decompressione dei file nell'archivio.

Gestione dell'estensione del file archiviato

Di default, il software utilizza l'estensione ".7bl" per i file archiviati, ad esempio, "archived_data.7bl". Tuttavia, gli utenti hanno la flessibilità di modificare l'estensione del file archiviato a loro piacimento, creando estensioni personalizzate o specificando il nome del file come preferiscono. L'importante è che il contenuto dell'archivio sarà crittografato e compresso indipendentemente dall'estensione del file.

Source Code - 7bl Compressor with Encryption

```
# Nome Software: 7bl Compressore con Crittografia
# Autore: Bocaletto Luca
# Descrizione: Questo software offre la possibilità di comprimere, crittografare ed estrarre file.
# Consente agli utenti di selezionare file da archiviare o da cui estrarre, specificare una chiave di crittografia,
# e generare archivi crittografati. È uno strumento utile per proteggere e comprimere i dati sensibili.
# Importa i moduli necessari

import tkinter as tk

from tkinter import ttk
from cryptography.fernet import Fernet
import zlib
import os
import traceback
from tkinter import filedialog
from cryptography.fernet import InvalidToken

# Definizione della classe principale dell'applicazione
```

```
class SevenBLCompressor:

    def __init__(self, window):

        self.window = window

        self.window.title("7bl Compressore con Crittografia")  # Imposta il titolo della finestra

        self.window.configure(bg="#f5f5f5")  # Imposta il colore di sfondo della finestra

        # Creazione e posizionamento degli elementi dell'interfaccia utente

        self.create_ui_elements()

    def create_ui_elements(self):

        self.file_label = tk.Label(self.window, text="7bl Compressore con Crittografia", font=("Helvetica", 20))

        self.file_label.pack()

        # Etichette, liste, campi di input, e pulsanti

        self.file_label = tk.Label(self.window, text="File da archiviare/estraire:")

        self.file_label.pack()

        self.file_listbox = tk.Listbox(self.window, selectmode=tk.MULTIPLE, width=50)

        self.file_listbox.pack()



        self.browse_input_button = tk.Button(self.window, text="Sfoglia", command=self.browse_input_files)

        self.browse_input_button.pack()

        self.output_label = tk.Label(self.window, text="File archiviato/estratto:")
```

```
        self.output_label.pack()

        self.output_entry = tk.Entry(self.window, width=50)

        self.output_entry.pack()

        self.browse_output_button = tk.Button(self.window, text="Sfoglia",
command=self.browse_output_directory)

        self.browse_output_button.pack()

        self.key_label = tk.Label(self.window, text="Chiave di crittografia:")

        self.key_label.pack()

        self.key_entry = tk.Entry(self.window, width=50)

        self.key_entry.pack()

        self.generate_key_button = tk.Button(self.window, text="Genera Chiave
Casuale", command=self.generate_and_set_key)

        self.generate_key_button.pack()

        self.progress_bar = ttk.Progressbar(self.window, orient="horizontal",
length=200, mode="determinate")

        self.progress_bar.pack()

        self.create_archive_button = tk.Button(self.window, text="Crea Archivio",
command=self.on_create_archive_button_click)

        self.create_archive_button.pack()



        self.extract_archive_button = tk.Button(self.window, text="Estrai
dall'Archivio", command=self.on_extract_archive_button_click)

        self.extract_archive_button.pack()

        self.result_label = tk.Label(self.window, text="")
```

```
    self.result_label.pack()
    # Personalizzazione dei colori dei pulsanti
    self.create_archive_button.config(bg="#4CAF50", fg="white")
    self.extract_archive_button.config(bg="#4CAF50", fg="white")
# Gestisce la selezione di file da archiviare/estraire
def browse_input_files(self):
    file_paths = filedialog.askopenfilenames()
    if file_paths:
        self.file_listbox.delete(0, tk.END)
        for file_path in file_paths:
            self.file_listbox.insert(tk.END, file_path)
# Gestisce la selezione della cartella di destinazione
def browse_output_directory(self):
    directory_path = filedialog.askdirectory()
    if directory_path:
        self.output_entry.delete(0, tk.END)
        self.output_entry.insert(0, directory_path)

# Genera una chiave casuale e la visualizza nell'interfaccia
def generate_key(self):
    return Fernet.generate_key()
def generate_and_set_key(self):
```

```
        key = self.generate_key()
        self.key_entry.delete(0, tk.END)
        self.key_entry.insert(0, key.decode())
    # Comprime e crittografa un file
    def compress_and_encrypt_file(self, input_file, output_file, key):
        try:
            with open(input_file, 'rb') as infile:
                data = infile.read()
                compressed_data = zlib.compress(data,
level=zlib.Z_BEST_COMPRESSION)
                encrypted_data = self.encrypt_data(compressed_data, key)
            with open(output_file, 'wb') as outfile:
                outfile.write(encrypted_data)
            self.update_progress_bar(100)  # Completato
        except Exception as e:
            traceback.print_exc()
            self.result_label.config(text="Errore durante la compressione e la
crittografia.")
            self.update_progress_bar(0)  # Errore

    # Decompatta e decrittografa un file
    def decrypt_and_decompress_file(self, input_file, output_file, key):
        try:
```

```
            with open(input_file, 'rb') as infile:
                encrypted_data = infile.read()
                decrypted_data = self.decrypt_data(encrypted_data, key)
                decompressed_data = zlib.decompress(decrypted_data)
            with open(output_file, 'wb') as outfile:
                outfile.write(decompressed_data)
            self.update_progress_bar(100)  # Completato
        except InvalidToken:
            self.result_label.config(text="Chiave di crittografia errata.")
            self.update_progress_bar(0)  # Errore
        except Exception as e:
            traceback.print_exc()
            self.result_label.config(text="Errore durante la decompressione e la decrittografia.")
            self.update_progress_bar(0)  # Errore
    # Crea un archivio contenente i file selezionati
    def create_archive(self, archive_file, files, key):
        archive_data = {}
        try:
            for file in files:
                with open(file, 'rb') as infile:
                    data = infile.read()
                    compressed_data = zlib.compress(data,
```

```
level=zlib.Z_BEST_COMPRESSION)
                encrypted_data = self.encrypt_data(compressed_data, key)
                archive_data[os.path.basename(file)] = encrypted_data
        with open(archive_file, 'wb') as outfile:
            outfile.write(str(archive_data).encode())
        self.update_status_message(f"Archivio {archive_file} creato!")
    except Exception as e:
        traceback.print_exc()
        self.result_label.config(text="Errore durante la creazione dell'archivio.")
        self.update_progress_bar(0)  # Errore
# Estrae i file dall'archivio
def extract_files_from_archive(self, archive_file, output_dir, key):
    try:
        with open(archive_file, 'rb') as infile:
            archive_data = eval(infile.read().decode())
        total_files = len(archive_data)
        for i, (filename, encrypted_data) in enumerate(archive_data.items()):
            self.update_status_message(f"Estrazione file {i+1}/{total_files}...")
            decrypted_data = self.decrypt_data(encrypted_data, key)
            decompressed_data = zlib.decompress(decrypted_data)
            output_file = os.path.join(output_dir, filename)
            with open(output_file, 'wb') as outfile:
```

```
                outfile.write(decompressed_data)
            self.update_progress_bar((i + 1) / total_files * 100)
        self.update_status_message("File estratti dall'archivio!")
    except InvalidToken:
        self.result_label.config(text="Chiave di crittografia errata.")
        self.update_progress_bar(0)  # Errore
    except Exception as e:
        traceback.print_exc()
        self.result_label.config(text="Errore durante l'estrazione dall'archivio.")
        self.update_progress_bar(0)  # Errore

# Crittografa i dati utilizzando una chiave
def encrypt_data(self, data, key):
    fernet = Fernet(key)
    return fernet.encrypt(data)
# Decrittografa i dati utilizzando una chiave
def decrypt_data(self, data, key):
    fernet = Fernet(key)
    return fernet.decrypt(data)

# Aggiorna la barra di avanzamento
def update_progress_bar(self, value):
```

```
    self.progress_bar["value"] = value
    self.window.update_idletasks()
# Aggiorna l'etichetta di stato
def update_status_message(self, message):
    self.result_label.config(text=message)
    self.window.update_idletasks()
# Gestisce il clic sul pulsante "Crea Archivio"
def on_create_archive_button_click(self):
    input_files = self.file_listbox.get(0, tk.END)
    output_dir = self.output_entry.get()
    key = self.key_entry.get().encode()
    if not input_files:
        self.result_label.config(text="Seleziona almeno un file da archiviare.")
        return
    if not output_dir:
        self.result_label.config(text="Seleziona una directory di destinazione.")
        return
    if not key:
        self.result_label.config(text="Inserisci una chiave di crittografia.")
        return
    for input_file in input_files:
        if not os.path.isfile(input_file):
```

```
                self.result_label.config(text=f"Il file {input_file} non esiste.")
                return
        try:
            archive_file = os.path.join(output_dir, "archived_data.7bl")
            self.update_status_message("Comprimendo e cifrando i file...")
            self.create_archive(archive_file, input_files, key)
        except Exception as e:
            return
        self.update_progress_bar(0)  # Reimposta la barra di avanzamento
        self.update_status_message("Archivio creato!")
    # Gestisce il clic sul pulsante "Estrai dall'Archivio"
    def on_extract_archive_button_click(self):
        archive_file = self.file_listbox.get(0)  # Si assume che sia selezionato solo un
file
        output_dir = self.output_entry.get()
        key = self.key_entry.get().encode()
        if not archive_file:
            self.result_label.config(text="Seleziona un archivio da cui estrarre.")
            return

        if not output_dir:
            self.result_label.config(text="Seleziona una directory di destinazione.")
            return
```

```
        if not key:
            self.result_label.config(text="Inserisci una chiave di crittografia.")
            return
        if not os.path.isfile(archive_file):
            self.result_label.config(text=f"L'archivio {archive_file} non esiste.")
            return
        if not os.path.exists(output_dir):
            os.mkdir(output_dir)
        try:
            self.update_status_message("Estrazione dall'archivio...")
            self.extract_files_from_archive(archive_file, output_dir, key)
        except Exception as e:
            return
        self.update_progress_bar(0)  # Reimposta la barra di avanzamento
        self.update_status_message("File estratti dall'archivio!")
# Punto di ingresso dell'applicazione
if __name__ == "__main__":
    window = tk.Tk()  # Crea una finestra principale
    app = SevenBLCompressor(window)  # Crea un'istanza dell'applicazione
    window.mainloop()  # Avvia il ciclo principale dell'interfaccia grafica
```

Morse Wave Translator

Autore: Bocaletto Luca

Descrizione

Il software "Morse Wave Translator" è un'applicazione progettata per registrare segnali audio e successivamente decodificarli in codice Morse. L'applicazione sfrutta alcune librerie Python per svolgere questa operazione, comprese PyQt5 per la creazione dell'interfaccia grafica, sounddevice per la registrazione audio, soundfile per la lettura e la scrittura dei file audio, pygame per la riproduzione audio e numpy per la manipolazione dei dati audio.

Componenti Principali

Il software è suddiviso in due classi principali:

1. **MorseDecoder:** Questa classe si occupa di decodificare i segnali audio registrati in codice Morse. Utilizza un dizionario che associa lettere e numeri ai rispettivi simboli del codice Morse. La decodifica si basa sulla lettura dell'ampiezza dei segnali audio: i picchi al di sopra di una soglia vengono considerati come "tratti" (dash), mentre i valori inferiori a tale soglia vengono considerati come "punti" (dot). La classe registra i segnali audio, li salva in un file audio e ne estrae il messaggio Morse decodificato.

2. **MorseDecoderApp:** Questa classe gestisce l'interfaccia grafica dell'applicazione. Consente all'utente di selezionare il dispositivo audio di input, specificare la durata della registrazione, registrare audio, decodificare il segnale Morse registrato, aprire file audio esistenti per la decodifica e riprodurre l'audio registrato. L'applicazione fornisce anche una guida per l'utente.

Source Code - Morse Wave Translator

```
# Software Name: Morse Wave Translator
# Author: Bocaletto Luca
# Descrizione: Questo software è un'applicazione che consente di registrare segnali audio e decodificarli in codice Morse.

# Import the sys module for accessing command line arguments.
import sys
# Import the PyQt5 module for creating a graphical interface.
import PyQt5.QtWidgets as QtWidgets
# Import the sounddevice module for audio recording.
import sounddevice as sd
# Import the numpy module for audio data handling.
import numpy as np
# Import the soundfile module for reading and writing audio files.
import soundfile as sf
# Import the os module for system operations.
import os
# Import the pygame module for audio playback.
```

```
import pygame
# Morse code dictionary mapping letters and numbers to Morse code representations.
MORSE_CODE_DICT = {
    'A': '.-', 'B': '-...', 'C': '-.-.', 'D': '-..', 'E': '.',
    'F': '..-.', 'G': '--.', 'H': '....', 'I': '..', 'J': '.---',
    'K': '-.-', 'L': '.-..', 'M': '--', 'N': '-.', 'O': '---',
    'P': '.--.', 'Q': '--.-', 'R': '.-.', 'S': '...', 'T': '-',
    'U': '..-', 'V': '...-', 'W': '.--', 'X': '-..-', 'Y': '-.--', 'Z': '--..',
    '1': '.----', '2': '..---', '3': '...--', '4': '....-', '5': '.....',
    '6': '-....', '7': '--...', '8': '---..', '9': '----.', '0': '-----'
}
# Class for Morse code decoding.
class MorseDecoder:
    def __init__(self):
                            # Initialize the sample rate for audio.
        self.sample_rate = 44100
        # Initialize an empty list to store audio data.
        self.audio = []
        # Initialize the pygame mixer for audio playback.
        pygame.mixer.init()

    def decode(self, audio_data):
                            # Initialize an empty string to store the decoded Morse code.
        decoded_text = ""
        # Implement Morse code decoding here.
        # Iterate through the audio data.
        for signal in audio_data:
            if signal > 0.5:  # Example: Check if the signal crosses a threshold.
                decoded_text += "-" # Append a dash for a "long" signal.
            else:
                decoded_text += "." # Append a dot for a "short" signal.
                            # Initialize an empty string to store the complete decoded
message.
        decoded_message = ""
        # Split the decoded text into individual Morse code signals.
        signal_list = decoded_text.split(" ")
        # Iterate through the list of Morse code signals.
        for signal in signal_list:
            if signal == "":
                decoded_message += " "  # Add a space to represent a word break.
            else:
                                                    # Find the corresponding character in the
Morse code dictionary.
                char = next((char for char, code in MORSE_CODE_DICT.items() if code
== signal), None)
```

```
            if char:
                decoded_message += char # Append the decoded character to the
message.
            else:
                decoded_message += "?" # If the signal is not recognized, use "?" as a
placeholder.
                        # Return the complete decoded message.
        return decoded_message

    def record_audio(self, duration, input_device_index=None):
                        # Record audio using the sounddevice library.
        # The audio is captured for the specified duration in seconds.
                        # Calculate the number of samples to record based on the sample
rate and duration.
        num_samples = int(self.sample_rate * duration)
                        # Use sounddevice to record audio with the specified parameters.
        self.audio = sd.rec(int(self.sample_rate * duration),
samplerate=self.sample_rate, channels=1, dtype='float32',
                            device=input_device_index)
        # Wait for the recording to complete.
        sd.wait()

    def save_audio(self, filename):
                        # Save recorded audio to a file using the soundfile library.
        # The audio is saved with the specified filename and sample rate.
        # Use soundfile library to write the recorded audio data to a file.
        sf.write(filename, self.audio, self.sample_rate)

    def decode_audio_file(self, audio_file):
                        # Read audio data from a file using the soundfile library.
        # Then, decode the audio data and return the decoded text.
        # Use soundfile library to read audio data from the specified file.
        audio_data, _ = sf.read(audio_file)
        # Decode the audio data using the MorseDecoder's decode method.
        decoded_text = self.decode(audio_data)
        # Return the decoded text.
        return decoded_text

    def play_audio(self, audio_data):
                        # Play audio using the pygame mixer.
        # Use the pygame mixer to play the specified audio data.
        pygame.mixer.Sound(audio_data).play()

class MorseDecoderApp(QtWidgets.QWidget):
    def __init__(self):
```

```
                    # Constructor for the MorseDecoderApp class.
        # Call the constructor of the parent class (QtWidgets.QWidget).
        super().__init__()
        # Initialize the list of input devices available.
        self.input_devices = self.get_input_devices()  # Inizializza la lista di periferiche
di input
        # Initialize the user interface (UI) for the application.
        self.init_ui()
        # Create an instance of the MorseDecoder class for Morse code processing.
        self.morse_decoder = MorseDecoder()

    def init_ui(self):
                    # Initialize the user interface (UI) elements of the application.
        # Set the title of the application window.
        self.setWindowTitle('Morse Wave Translator')
        # Set the initial position and size of the application window.
        self.setGeometry(100, 100, 600, 300)
                    # Create a QPushButton widget with the label "Record."
        self.record_button = QtWidgets.QPushButton('Record')
        # Set an icon for the "Record" button, using the standard media play icon from
the QApplication style.

self.record_button.setIcon(QtWidgets.QApplication.style().standardIcon(QtWidgets.
QStyle.SP_MediaPlay))
        # Connect the "clicked" signal of the "Record" button to the "record_audio"
method.
        self.record_button.clicked.connect(self.record_audio)

        self.decode_button = QtWidgets.QPushButton('Decode')

self.decode_button.setIcon(QtWidgets.QApplication.style().standardIcon(QtWidgets.
QStyle.SP_MediaSeekForward))
        self.decode_button.clicked.connect(self.decode_audio)

        self.open_file_button = QtWidgets.QPushButton('Apri File')

self.open_file_button.setIcon(QtWidgets.QApplication.style().standardIcon(QtWidge
ts.QStyle.SP_DirOpenIcon))
        self.open_file_button.clicked.connect(self.open_audio_file)

        self.duration_input = QtWidgets.QLineEdit(self)
        self.duration_input.setPlaceholderText("Durata della registrazione (secondi)")

        self.input_device_label = QtWidgets.QLabel("Dispositivo di Input:")
        self.input_device_combo = QtWidgets.QComboBox(self)
```

```
        self.input_device_combo.addItems(self.input_devices)  # Imposta la lista di periferiche di input

        self.duration_label = QtWidgets.QLabel("Durata della registrazione:")
        self.text_output_label = QtWidgets.QLabel("Risultati:")

        self.text_output = QtWidgets.QTextEdit()
        self.text_output.setReadOnly(True)

        self.help_button = QtWidgets.QPushButton('Guida')
        self.help_button.clicked.connect(self.show_help)

        self.play_button = QtWidgets.QPushButton('Play')

self.play_button.setIcon(QtWidgets.QApplication.style().standardIcon(QtWidgets.QStyle.SP_MediaPlay))
        self.play_button.clicked.connect(self.play_recorded_audio)

        layout = QtWidgets.QVBoxLayout()
        layout.addWidget(self.input_device_label)
        layout.addWidget(self.input_device_combo)
        layout.addWidget(self.duration_label)
        layout.addWidget(self.duration_input)
        layout.addWidget(self.record_button)
        layout.addWidget(self.decode_button)
        layout.addWidget(self.play_button)
        layout.addWidget(self.open_file_button)
        layout.addWidget(self.text_output_label)
        layout.addWidget(self.text_output)
        layout.addWidget(self.help_button)

        self.setLayout(layout)

    def record_audio(self):
        duration_str = self.duration_input.text()
        try:
            duration = float(duration_str)
        except ValueError:
            self.text_output.append("Errore: Inserisci una durata valida.")
            return

        if duration <= 0:
            self.text_output.append("Errore: La durata deve essere maggiore di zero.")
            return
```

```
        input_device_index = self.input_device_combo.currentIndex()
        try:
            self.morse_decoder.record_audio(duration, input_device_index)
        except sd.PortAudioError as e:
            self.text_output.append(f"Errore durante la registrazione: {str(e)}")
            return
        except Exception as e:
            self.text_output.append(f"Errore sconosciuto durante la registrazione:
{str(e)}")
            return

        self.text_output.clear()
        self.text_output.append(f"Audio registrato per {duration} secondi")

    def decode_audio(self):
        if len(self.morse_decoder.audio) > 0:
            try:
                self.morse_decoder.save_audio('recorded_audio.wav')
                audio_data, _ = sf.read('recorded_audio.wav')
                decoded_text = self.morse_decoder.decode(audio_data)
                self.text_output.append(f"Decoded Morse Code: {decoded_text}")
            except sf.SoundFileError as e:
                self.text_output.append(f"Errore durante la decodifica del file audio:
{str(e)}")
            except Exception as e:
                self.text_output.append(f"Errore sconosciuto durante la decodifica:
{str(e)}")
        else:
            self.text_output.append("Nessun audio registrato.")

    def open_audio_file(self):
        options = QtWidgets.QFileDialog.Options()
        file_name, _ = QtWidgets.QFileDialog.getOpenFileName(self, "Apri File
Audio", "", "File Audio (*.wav);;Tutti i Files (*)", options=options)
        if file_name:
            try:
                decoded_text = self.morse_decoder.decode_audio_file(file_name)
                self.text_output.append(f"Decoded Morse Code from File:
{decoded_text}")
            except sf.SoundFileError as e:
                self.text_output.append(f"Errore durante l'apertura del file audio: {str(e)}")
            except Exception as e:
                self.text_output.append(f"Errore sconosciuto durante l'apertura del file
audio: {str(e)}")
```

```
    def play_recorded_audio(self):
        if len(self.morse_decoder.audio) > 0:
            try:
                self.morse_decoder.save_audio('playback_audio.wav')
                audio_data, _ = sf.read('playback_audio.wav')
                self.morse_decoder.play_audio(audio_data)
            except sf.SoundFileError as e:
                self.text_output.append(f"Errore durante la riproduzione del file audio: {str(e)}")
            except Exception as e:
                self.text_output.append(f"Errore sconosciuto durante la riproduzione del file audio: {str(e)}")
        else:
            self.text_output.append("Nessun audio registrato per la riproduzione.")

    def show_help(self):
        QtWidgets.QMessageBox.information(self, "Guida",
        "Questa applicazione consente di registrare segnali audio e decodificarli in codice Morse. Ecco come utilizzarla:\n\n"
        "1. **Selezione del Dispositivo di Input:** Nel menu a discesa 'Dispositivo di Input', puoi selezionare il dispositivo audio di input che desideri utilizzare per la registrazione. Assicurati che il tuo dispositivo audio sia correttamente configurato.\n\n"
        "2. **Registrazione:** Nel campo 'Durata della registrazione (secondi)', inserisci la durata desiderata per la registrazione e premi il pulsante 'Record'. L'app registrerà l'audio dal dispositivo selezionato per il periodo specificato.\n\n"
        "3. **Decodifica:** Dopo la registrazione, premi il pulsante 'Decode' per decodificare il segnale Morse registrato. Il risultato della decodifica verrà mostrato nell'area 'Risultati'.\n\n"
        "4. **Apertura di File Audio:** Se hai un file audio con segnali Morse, puoi aprirlo premendo il pulsante 'Apri File'. L'app decodificherà il contenuto del file e lo mostrerà nell'area 'Risultati'.\n\n"
        "5. **Riproduzione Audio Registrato:** Dopo la registrazione, premendo il pulsante 'Play', puoi ascoltare nuovamente l'audio registrato.\n\n"
        "Assicurati di aver selezionato il dispositivo di input corretto e di impostare una durata di registrazione adeguata. Buon divertimento decodificando segnali Morse!\n\n"
        "Autore: Bocaletto Luca Aka Elektronoide\n\n"
        "Sito Web: https://www.elektronoide.it")

    def get_input_devices(self):
        input_devices = []
        devices = sd.query_devices()
        for i, device in enumerate(devices):
            if 'input' in device['name'].lower():
```

```
                input_devices.append(f"{i}: {device['name']}")
        return input_devices

if __name__ == '__main__':
    app = QtWidgets.QApplication(sys.argv)
    window = MorseDecoderApp()
    window.show()
    sys.exit(app.exec_())
```

Generatore CW (Morse)

Autore: Bocaletto Luca

Descrizione

Il Generatore CW (Morse) è un'applicazione versatile e potente per gli appassionati di comunicazioni Morse e per chiunque sia interessato a imparare o praticare questo classico linguaggio di comunicazione. Questo software ti consente di convertire il testo in codice Morse e viceversa, fornendo una suite completa di strumenti per creare, interpretare e riprodurre segnali Morse con facilità.

Caratteristiche principali

- **Conversione Testo-Morse e Morse-Testo**
    - Da Testo a Morse: Inserisci il testo desiderato e il software genererà immediatamente il corrispondente codice Morse. Puoi personalizzare la velocità dei beep e il numero di ripetizioni per l'audio Morse.
    - Da Morse a Testo: Inserisci il codice Morse e il software lo tradurrà istantaneamente in testo leggibile, rendendo più semplice che mai decifrare i segnali Morse.
- **Riproduzione Audio Morse**
    - Sperimenta l'esperienza autentica di ascoltare i segnali Morse con la funzionalità di riproduzione audio. Il software emette beep per i punti e i trattini, consentendoti di acquisire familiarità con il suono distintivo del codice Morse.
- **Interfaccia Utente Intuitiva**
    - L'interfaccia utente basata su Tkinter è user-friendly e facile da navigare. Puoi inserire il tuo testo o codice Morse, visualizzare le traduzioni e controllare la generazione dell'audio Morse con un solo clic.
- **Personalizzazione**
    - Regola la velocità dei beep e il numero di ripetizioni audio per adattare l'applicazione alle tue preferenze e al tuo stile di apprendimento.

Come Iniziare

- **Da Testo a Morse:** Inserisci il testo nell'apposito campo e personalizza la velocità e le ripetizioni. Clicca su "Da Testo a Morse" per generare il codice Morse.
- **Da Morse a Testo:** Inserisci il codice Morse e clicca su "Da Morse a Testo" per ottenere la traduzione in testo.
- **Riproduzione Audio:** Utilizza la funzione "Riproduzione Audio" per ascoltare i segnali Morse generati.

Personalizzazione Avanzata

- **Velocità dei Beep:** Regola la velocità dei beep per adattarli alle tue preferenze.

- **Ripetizioni Audio:** Scegli quante volte ripetere l'audio Morse.

Questo software è un'ottima risorsa per gli appassionati di radioamatori, operatori di comunicazioni di emergenza, apprendisti Morse e tutti coloro che desiderano esplorare il linguaggio Morse in modo interattivo.

Source Code – Generatore CW Morse

```
# Software Name: Generatore CW (Morse)
# Author: Bocaletto Luca
# Description: Questo programma genera e riproduce segnali CW (Morse) da testo e viceversa.

# Importa i moduli necessari
import tkinter as tk
import winsound
import threading  # Importa il modulo threading per gestire il thread di generazione audio

# Definizione delle costanti per la durata dei punti e dei trattini
DOT_DURATION = 100  # Durata in millisecondi di un punto
DASH_DURATION = 300  # Durata in millisecondi di un trattino

# Definizione del codice Morse
morse_code = {
    'A': '.-', 'B': '-...', 'C': '-.-.', 'D': '-..', 'E': '.',
    'F': '..-.', 'G': '--.', 'H': '....', 'I': '..', 'J': '.---',
    'K': '-.-', 'L': '.-..', 'M': '--', 'N': '-.', 'O': '---',
    'P': '.--.', 'Q': '--.-', 'R': '.-.', 'S': '...', 'T': '-',
    'U': '..-', 'V': '...-', 'W': '.--', 'X': '-..-', 'Y': '-.--', 'Z': '--..',
    '0': '-----', '1': '.----', '2': '..---', '3': '...--', '4': '....-', '5': '.....',
    '6': '-....', '7': '--...', '8': '---..', '9': '----.',
    ' ': ' ', ',': '--..--', '.': '..-.-.-', '?': '..--..',
}

# Variabile globale per il thread di generazione audio
audio_thread = None

# Funzione per generare il segnale CW e l'audio Morse
def generate_cw(text, speed, repetitions):
    global audio_thread  # Utilizza la variabile globale per il thread audio
    morse_text = ""
    audio_morse = ""
    for char in text:
```

```
        if char.isalpha():
            morse_char = char.upper()
            if morse_char in morse_code:
                morse_sequence = morse_code[morse_char]
                for symbol in morse_sequence:
                    if symbol == '.':
                        morse_text += "."  # Aggiungi un punto al codice Morse
                        audio_morse += "."  # Aggiungi un punto al suono Morse
                    elif symbol == '-':
                        morse_text += "-"  # Aggiungi un trattino al codice Morse
                        audio_morse += "-"  # Aggiungi un trattino al suono Morse
                morse_text += " "  # Aggiungi uno spazio tra i caratteri Morse
                audio_morse += " "  # Aggiungi uno spazio tra i caratteri Morse
    morse_text = morse_text.strip()
    morse_label.config(text="Conversione in Codice Morse: " + morse_text)
    audio_morse = audio_morse.replace(" ", "")  # Rimuovi gli spazi

    # Funzione per generare audio Morse in un thread separato
    def play_audio():
        for _ in range(repetitions):
            if audio_thread is None:
                return  # Interrompi la generazione se il thread è nullo
            for char in audio_morse:
                if char == '.':
                    winsound.Beep(800, int(DOT_DURATION * speed))  # Riproduci un
punto
                elif char == '-':
                    winsound.Beep(800, int(DASH_DURATION * speed))  # Riproduci un
trattino

    audio_thread = StoppableThread(target=play_audio)
    audio_thread.start()  # Avvia il thread audio

# Funzione per fermare la generazione audio
def stop_audio_generation():
    global audio_thread
    if audio_thread is not None:
        audio_thread.stop()  # Chiudi il thread audio se esiste
        audio_thread = None  # Imposta il thread audio su None

# Funzione per convertire da Morse a testo e generare l'audio Morse
def morse_to_text_conversion():
    morse_text = morse_entry.get()
    text = ""
    for morse_word in morse_text.split(" "):
```

```
        for morse_char in morse_word.split(" "):
            for key, value in morse_code.items():
                if value == morse_char:
                    text += key
                    for symbol in morse_char:
                        if symbol == '.':
                            winsound.Beep(800, int(DOT_DURATION))  # Riproduci un punto
                        elif symbol == '-':
                            winsound.Beep(800, int(DASH_DURATION))  # Riproduci un
trattino
            text += " "  # Aggiungi uno spazio tra le parole
    text_entry.delete(0, tk.END)
    text_entry.insert(0, text)

# Classe Thread personalizzata che può essere interrotta
class StoppableThread(threading.Thread):
    def __init__(self, *args, **kwargs):
        super(StoppableThread, self).__init__(*args, **kwargs)
        self._stop_event = threading.Event()

    def stop(self):
        self._stop_event.set()

    def stopped(self):
        return self._stop_event.is_set()

# Creazione della GUI
root = tk.Tk()
root.title("Generatore CW (Morse)")

text_label = tk.Label(root, text="Inserisci il Testo da convertire:")
text_label.pack()

text_entry = tk.Entry(root, width=40)
text_entry.pack()

text_label = tk.Label(root, text="Inserisci il Codice Morse da convertire:")
text_label.pack()

morse_entry = tk.Entry(root, width=40)
morse_entry.pack()

morse_label = tk.Label(root, text="Conversione in Morse:")
morse_label.pack()
```

```
text_to_morse_button = tk.Button(root, text="Da Testo a Morse", command=lambda: generate_cw(text_entry.get(), speed_scale.get(), repetitions_scale.get()))
text_to_morse_button.pack()

morse_to_text_button = tk.Button(root, text="Da Morse a Testo", command=morse_to_text_conversion)
morse_to_text_button.pack()

stop_button = tk.Button(root, text="Arresta Generazione", command=stop_audio_generation)
stop_button.pack()

repetitions_label = tk.Label(root, text="Ripetizioni Audio:")
repetitions_label.pack()
repetitions_scale = tk.Scale(root, from_=1, to=10, orient="horizontal")
repetitions_scale.pack()

speed_label = tk.Label(root, text="Velocità dei Beep:")
speed_label.pack()
speed_scale = tk.Scale(root, from_=0.1, to=2, orient="horizontal", resolution=0.1)
speed_scale.set(1.0)  # Imposta il valore predefinito a 1.0
speed_scale.pack()

root.mainloop()
```

Reverb - Applicazione per Effetti Audio di Riverbero

Descrizione

"Reverb" è un'applicazione Python sviluppata da Luca Bocaletto che consente di creare l'effetto audio di riverbero. Questa applicazione offre agli utenti la possibilità di aggiungere riverbero alle registrazioni audio, creando un suono spaziale e riflessivo che simula l'eco di un suono all'interno di uno spazio fisico.

Funzionalità Principali

- **Regolazione dei Parametri:** Gli utenti possono regolare i seguenti parametri dell'effetto di riverbero:
    - Guadagno del Riverbero: Il livello di aumento del segnale di riverbero rispetto all'audio originale.
    - Lunghezza del Riverbero: La durata del riverbero, misurata in millisecondi, che influenza quanto tempo il suono continua a riflettersi.
    - Frequenza di Taglio: La frequenza di taglio del riverbero, che controlla la

distribuzione delle frequenze nel suono riverberato.
        - Fattore di Decay: Il tasso di decadimento del riverbero, che determina quanto velocemente l'eco si attenua nel tempo.
- **Attivazione/Disattivazione:** L'applicazione consente agli utenti di attivare o disattivare l'effetto di riverbero con un semplice pulsante.
- **Interfaccia Utente Intuitiva:** L'interfaccia utente è progettata in modo semplice e intuitivo, con slider per ogni parametro e una visualizzazione chiara dei valori attuali.
- **Elaborazione Audio in Tempo Reale:** L'applicazione elabora l'audio in tempo reale, consentendo agli utenti di ascoltare immediatamente l'effetto di riverbero applicato.

Source Code – Reverbero Audio Effetto

```
# Nome del Software: Reverb
# Autore: Luca Bocaletto
# Descrizione: Applicazione Python per creare un effetto audio Reverb.

import sys
import numpy as np
import sounddevice as sd
from PyQt5.QtWidgets import QApplication, QWidget, QVBoxLayout, QSlider,
QLabel, QPushButton
from scipy.signal import lfilter
from PyQt5.QtCore import Qt
from PyQt5.QtGui import QFont

# Definizione delle costanti
FREQUENZA_CAMPIONAMENTO = 44100
DIMENSIONE_BLOCCO = 8192

class AppReverb(QWidget):
    def __init__(self):
        super().__init__()
        self.inizializza_predefiniti()
        self.inizializza_UI()
        self.inizializza_Audio()
        self.reverb_attivo = False

    def inizializza_predefiniti(self):
        # Inizializza i parametri predefiniti
        self.gain_reverb = 0.8  # Aumento del guadagno del riverbero
        self.risposta_impulso = None
        self.lunghezza_reverb = 3000
        self.frequenza_di_taglio = 0.5
        self.fattore_decay = 0.8
```

```
        self.buffer_audio = np.zeros(128)
        self.slider = []

    def inizializza_UI(self):
        # Inizializza l'interfaccia utente
        self.setGeometry(100, 100, 400, 400)
        self.setWindowTitle('Reverbero')
        self.setStyleSheet("background-color: #4B0082; color: yellow;")
        self.label_titolo = QLabel('Reverbero')
        self.label_titolo.setAlignment(Qt.AlignCenter)
        self.label_titolo.setFont(QFont('Arial', 24))

        self.slider_reverb = self.crea_slider(self.gain_reverb,
self.aggiorna_guadagno_reverb, "Guadagno Reverbero: {:.2f}")
        self.slider_lunghezza_reverb = self.crea_slider(self.lunghezza_reverb / 10,
self.aggiorna_lunghezza_reverb, "Lunghezza Reverbero: {}")
        self.slider_frequenza_di_taglio = self.crea_slider(self.frequenza_di_taglio,
self.aggiorna_frequenza_di_taglio, "Frequenza di Taglio: {:.2f}")
        self.slider_fattore_decay = self.crea_slider(self.fattore_decay,
self.aggiorna_fattore_decay, "Fattore di Decay: {:.2f}")

        self.pulsante_toggle = QPushButton('Attiva/Disattiva Reverbero')
        self.pulsante_toggle.clicked.connect(self.cambia_stato_reverb)

        layout = QVBoxLayout()
        layout.addWidget(self.label_titolo)
        for slider, label in self.slider:
            layout.addWidget(label)
            layout.addWidget(slider)
        layout.addWidget(self.pulsante_toggle)
        self.setLayout(layout)

    def crea_slider(self, valore, callback, formato_label):
        # Crea e configura uno slider
        slider = QSlider(orientation=1)
        slider.setRange(0, 100)
        slider.setValue(int(valore * 100) if isinstance(valore, float) else valore)
        label = QLabel(formato_label.format(valore))
        slider.valueChanged.connect(callback)
        self.slider.append((slider, label))
        return slider

    def inizializza_Audio(self):
        def callback_audio(indata, outdata, frames, time, status):
            if status:
```

```
            print("Stato Audio:", status, file=sys.stderr)

        if self.reverb_attivo and self.risposta_impulso is not None:
            # Applica l'effetto di riverbero all'input audio
            riverbero = lfilter(self.risposta_impulso, [1, -self.gain_reverb], indata)
            outdata[:] = riverbero
        else:
            outdata[:] = indata

    self.stream = sd.Stream(
        callback=callback_audio,
        samplerate=FREQUENZA_CAMPIONAMENTO,
        channels=2,
        blocksize=DIMENSIONE_BLOCCO
    )

    self.stream.start()

def aggiorna_guadagno_reverb(self):
    # Aggiorna il guadagno del riverbero in base all'input dello slider
    self.gain_reverb = self.slider_reverb.value() / 100.0
    self.slider[0][1].setText(f"Guadagno Reverbero: {self.gain_reverb:.2f}")

def aggiorna_lunghezza_reverb(self):
    # Aggiorna la lunghezza del riverbero in base all'input dello slider
    self.lunghezza_reverb = int(self.slider_lunghezza_reverb.value() * 10)
    self.aggiorna_risposta_impulso()
    self.slider[1][1].setText(f"Lunghezza Reverbero: {self.lunghezza_reverb}")

def aggiorna_frequenza_di_taglio(self):
    # Aggiorna la frequenza di taglio in base all'input dello slider
    self.frequenza_di_taglio = self.slider_frequenza_di_taglio.value() / 100.0
    self.slider[2][1].setText(f"Frequenza di Taglio: {self.frequenza_di_taglio:.2f}")

def aggiorna_fattore_decay(self):
    # Aggiorna il fattore di decay in base all'input dello slider
    self.fattore_decay = self.slider_fattore_decay.value() / 100.0
    self.slider[3][1].setText(f"Fattore di Decay: {self.fattore_decay:.2f}")

def aggiorna_risposta_impulso(self):
    if self.lunghezza_reverb > 0:
        # Genera una risposta all'impulso casuale
        self.risposta_impulso = np.random.randn(self.lunghezza_reverb)
        self.risposta_impulso = np.convolve(self.risposta_impulso, [1,
self.fattore_decay], mode='full')
```

```
            self.risposta_impulso /= np.max(np.abs(self.risposta_impulso))
        else:
            self.risposta_impulso = None

    def cambia_stato_reverb(self):
        # Attiva/Disattiva l'effetto di riverbero
        self.reverb_attivo = not self.reverb_attivo

if __name__ == '__main__':
    app = QApplication(sys.argv)
    ex = AppReverb()
    ex.show()
    sys.exit(app.exec_())
```

Chorus - Applicazione per Effetti Audio Chorus

Descrizione

"Chorus" è un'applicazione in Python sviluppata da Luca Bocaletto che consente di creare l'effetto

audio Chorus. Questa applicazione offre agli utenti la possibilità di regolare diversi parametri dell'effetto Chorus per modificare l'audio in modo creativo. L'effetto Chorus è ampiamente utilizzato in produzioni audio per aggiungere profondità e spazialità al suono, creando l'impressione di più suonatori o cantanti che eseguono simultaneamente.

Funzionalità Principali

- **Regolazione dei Parametri:** Gli utenti possono regolare i seguenti parametri dell'effetto Chorus:
    - Ritardo Chorus: Il ritardo tra le repliche dell'audio, misurato in millisecondi.
    - Intensità: L'intensità dell'effetto Chorus, che controlla quanto sia evidente l'effetto.
    - Tasso LFO (Oscillatore a Bassa Frequenza): Il tasso di modulazione dell'effetto Chorus, che influisce sulla velocità con cui il suono oscilla.
    - Feedback: Il feedback determina quanto dell'audio elaborato viene reinserito nel processo di Chorus, influenzando la persistenza delle repliche sonore.
    - Mischiamento: Il mischiamento tra l'audio originale e l'audio Chorus elaborato.
- **Attivazione/Disattivazione:** L'applicazione consente agli utenti di attivare o disattivare l'effetto Chorus con un semplice pulsante.
- **Interfaccia Utente Intuitiva:** L'interfaccia utente è progettata in modo semplice e intuitivo, con slider per ogni parametro e una visualizzazione chiara dei valori attuali.
- **Elaborazione Audio in Tempo Reale:** L'applicazione elabora l'audio in tempo reale e consente agli utenti di ascoltare immediatamente l'effetto Chorus.

Utilizzo

1. Regola i parametri dell'effetto Chorus utilizzando gli slider corrispondenti per ottenere il suono desiderato.
2. Premi il pulsante "Attiva/Disattiva Chorus" per abilitare o disabilitare l'effetto Chorus.
3. Ascolta l'audio con l'effetto Chorus applicato in tempo reale.

"Chorus" è uno strumento utile per musicisti, produttori musicali e appassionati di audio che desiderano sperimentare con l'effetto Chorus e aggiungere profondità e risonanza alle loro registrazioni audio.

```
# Nome del Software: Chorus
# Autore: Luca Bocaletto
# Descrizione: Applicazione Python per creare un effetto audio Chorus.

# Importa le librerie necessarie
import sys
import numpy as np
import sounddevice as sd
from PyQt5.QtCore import Qt
from PyQt5.QtGui import QFont
from PyQt5.QtWidgets import QApplication, QWidget, QVBoxLayout, QSlider,
```

```
QLabel, QPushButton

# Definizione delle costanti
FREQUENZA_CAMPIONAMENTO = 44100  # Frequenza di campionamento per l'elaborazione audio
DIMENSIONE_BLOCCO = 8192  # Dimensione del blocco di elaborazione audio

# Crea un'applicazione basata su QWidget per l'effetto Chorus
class AppChorus(QWidget):
    def __init__(self):
        super().__init__()
        self.inizializza_predefiniti()  # Inizializza le impostazioni predefinite
        self.inizializza_UI()  # Inizializza l'interfaccia utente
        self.inizializza_Audio()  # Inizializza l'elaborazione audio
        self.is_chorus_enabled = False  # Flag per attivare/disattivare l'effetto Chorus
        self.buffer_audio = np.zeros(DIMENSIONE_BLOCCO)  # Inizializza un buffer audio

    def inizializza_predefiniti(self):
        # Parametri predefiniti per l'effetto Chorus
        self.ritardo = 0.025
        self.intensita = 0.5
        self.tasso_lfo = 0.5
        self.feedback = 0.3
        self.mischiamento = 0.5
        # Crea un buffer di ritardo per l'elaborazione audio
        self.buffer_ritardo = np.zeros(int(self.ritardo * FREQUENZA_CAMPIONAMENTO) + DIMENSIONE_BLOCCO)
        self.fase_lfo = 0

        self.slider = []  # Lista per memorizzare gli elementi dello slider

    def inizializza_UI(self):
        self.setGeometry(100, 100, 400, 400)  # Imposta le dimensioni della finestra
        self.setWindowTitle('Chorus')  # Imposta il titolo della finestra
        self.setStyleSheet("background-color: #4B0082; color: yellow;")  # Definisci lo stile della finestra

        # Crea un QLabel per visualizzare il titolo
        self.label_titolo = QLabel('Chorus')
        self.label_titolo.setAlignment(Qt.AlignCenter)  # Imposta l'allineamento del testo
        self.label_titolo.setFont(QFont('Arial', 24))  # Imposta il carattere del testo

        # Crea gli slider per regolare i parametri del Chorus
```

```
        self.slider_ritardo = self.crea_slider(self.ritardo, self.aggiornaRitardo, "Ritardo
Chorus: {:.2f} ms")
        self.slider_intensita = self.crea_slider(self.intensita, self.aggiornaIntensita,
"Intensità: {:.2f}")
        self.slider_tasso_lfo = self.crea_slider(self.tasso_lfo, self.aggiornaTassoLFO,
"Tasso LFO: {:.2f}")
        self.slider_feedback = self.crea_slider(self.feedback, self.aggiornaFeedback,
"Feedback: {:.2f}")
        self.slider_mischiamento = self.crea_slider(self.mischiamento,
self.aggiornaMischiamento, "Mischiamento: {:.2f}")

        # Crea un pulsante per attivare/disattivare l'effetto Chorus
        self.pulsante_toggle = QPushButton('Attiva/Disattiva Chorus')
        self.pulsante_toggle.clicked.connect(self.cambiaStatoChorus)

        layout = QVBoxLayout()  # Crea un layout per i widget
        layout.addWidget(self.label_titolo)  # Aggiungi il label del titolo al layout

        # Aggiungi gli slider e i label al layout
        for slider, label in self.slider:
            layout.addWidget(label)
            layout.addWidget(slider)

        layout.addWidget(self.pulsante_toggle)  # Aggiungi il pulsante di attivazione al
layout

        self.setLayout(layout)  # Imposta il layout per la finestra principale

    def crea_slider(self, valore, callback, formato_label):
        slider = QSlider(orientation=1)  # Crea un elemento slider
        slider.setRange(0, 100)  # Imposta il range dello slider
        slider.setValue(int(valore * 100) if isinstance(valore, float) else valore)  #
Imposta il valore iniziale

        label = QLabel(formato_label.format(valore))  # Crea un label per visualizzare il
valore del parametro

        slider.valueChanged.connect(callback)  # Collega lo slider alla sua funzione di
callback

        self.slider.append((slider, label))  # Aggiungi lo slider e il label alla lista

        return slider

    def inizializza_Audio(self):
```

```
        def callback_audio(indata, outdata, frames, time, status):
            if status:
                print("Stato Audio:", status, file=sys.stderr)

            if self.is_chorus_enabled:
                offset = int(self.intensita * DIMENSIONE_BLOCCO * np.sin(2 * np.pi *
self.fase_lfo))
                if offset >= len(self.buffer_ritardo):
                    offset = len(self.buffer_ritardo) - 1

                # Copia i campioni di input nel buffer di ritardo
                self.buffer_ritardo[:DIMENSIONE_BLOCCO] = indata[:, 0] +
self.feedback * self.buffer_ritardo[:DIMENSIONE_BLOCCO]

                # Applica l'effetto Chorus mescolando l'audio in ingresso e l'audio ritardato
                bagnato = self.mischiamento * indata[:, 0] + (1 - self.mischiamento) *
self.buffer_ritardo[:DIMENSIONE_BLOCCO]
                outdata[:, 0] = bagnato
                outdata[:, 1] = bagnato

                # Aggiorna la fase dell'Oscillatore a Bassa Frequenza (LFO)
                self.fase_lfo += self.tasso_lfo * DIMENSIONE_BLOCCO /
FREQUENZA_CAMPIONAMENTO

            else:
                outdata[:] = indata

        # Crea uno stream audio per l'elaborazione dei dati audio
        self.stream = sd.Stream(
            callback=callback_audio,
            samplerate=FREQUENZA_CAMPIONAMENTO,
            channels=2,
            blocksize=DIMENSIONE_BLOCCO
        )

        self.stream.start()

    def aggiornaRitardo(self):
        self.ritardo = self.slider_ritardo.value() / 1000.0
        self.buffer_ritardo = np.zeros(int(self.ritardo *
FREQUENZA_CAMPIONAMENTO) + DIMENSIONE_BLOCCO)
        self.slider[0][1].setText(f"Ritardo Chorus: {self.ritardo * 1000:.2f} ms")

    def aggiornaIntensita(self):
        self.intensita = self.slider_intensita.value() / 100.0
```

```
        self.slider[1][1].setText(f"Intensità: {self.intensita:.2f}")

    def aggiornaTassoLFO(self):
        self.tasso_lfo = self.slider_tasso_lfo.value() / 100.0
        self.slider[2][1].setText(f"Tasso LFO: {self.tasso_lfo:.2f}")

    def aggiornaFeedback(self):
        self.feedback = self.slider_feedback.value() / 100.0
        self.slider[3][1].setText(f"Feedback: {self.feedback:.2f}")

    def aggiornaMischiamento(self):
        self.mischiamento = self.slider_mischiamento.value() / 100.0
        self.slider[4][1].setText(f"Mischiamento: {self.mischiamento:.2f}")

    def cambiaStatoChorus(self):
        self.is_chorus_enabled = not self.is_chorus_enabled

# Punto di ingresso dell'applicazione
if __name__ == '__main__':
    app = QApplication(sys.argv)
    ex = AppChorus()
    ex.show()
    sys.exit(app.exec_())
```

PyPI Search Library

Descrizione

PyPI Search Library è un'applicazione Python che ti consente di cercare librerie Python su PyPI (Python Package Index) in modo rapido e semplice. Questa libreria è stata sviluppata da Bocaletto Luca e fornisce un'interfaccia utente intuitiva per effettuare ricerche nel vasto e ricco ecosistema di librerie Python disponibili su PyPI. Gli utenti possono inserire il nome della libreria che desiderano cercare, fare clic sul pulsante "Cerca" e visualizzare l'elenco delle librerie corrispondenti.

Caratteristiche Principali

- **Interfaccia Utente Intuitiva**: PyPI Search Library offre un'interfaccia utente user-friendly che consente agli utenti di inserire facilmente il nome delle librerie da cercare.
- **Ricerca Veloce**: Grazie all'integrazione con PyPI, l'applicazione è in grado di effettuare ricerche veloci ed efficienti, restituendo risultati in tempo reale.
- **Visualizzazione dei Risultati**: I risultati della ricerca vengono presentati in una lista chiara e organizzata, consentendo agli utenti di visualizzare facilmente le librerie corrispondenti al proprio interesse.
- **Semplice da Usare**: L'applicazione è stata progettata per essere accessibile anche a utenti senza esperienza tecnica avanzata, offrendo un modo semplice per esplorare il vasto mondo delle librerie Python.
- **Open Source**: PyPI Search Library è un software open-source, il che significa che è possibile esaminare il codice sorgente, contribuire al suo sviluppo o personalizzarlo per le proprie esigenze.

Source Code - PyPI Search Library

```
# Nome del Software: PyPI Search Library
# Autore: Bocaletto Luca
import sys
import requests
from PyQt5.QtWidgets import QApplication, QMainWindow, QVBoxLayout,
QWidget, QPushButton, QLineEdit, QListWidget, QListWidgetItem

class PyPIBrowser(QMainWindow):
    def __init__(self):
        super().__init__()

        # Impostazioni della finestra principale
        self.setWindowTitle("PyPI Search Library")  # Imposta il titolo della finestra
```

```
        self.setGeometry(100, 100, 800, 600)  # Imposta la posizione e le dimensioni della finestra

        # Creazione del widget centrale
        self.central_widget = QWidget()
        self.setCentralWidget(self.central_widget)  # Imposta il widget centrale per la finestra

        # Creazione del layout verticale
        self.layout = QVBoxLayout()
        self.central_widget.setLayout(self.layout)  # Imposta il layout come layout centrale

        # Campo di input per la ricerca
        self.search_input = QLineEdit(self)
        self.search_input.setPlaceholderText("Cerca librerie su PyPI")  # Imposta un testo di esempio nel campo di input
        self.layout.addWidget(self.search_input)  # Aggiunge il campo di input al layout

        # Pulsante di ricerca
        self.search_button = QPushButton("Cerca", self)
        self.layout.addWidget(self.search_button)  # Aggiunge il pulsante di ricerca al layout

        # Lista per i risultati
        self.result_list = QListWidget(self)
        self.layout.addWidget(self.result_list)  # Aggiunge la lista al layout

        # Collega il clic del pulsante all'azione di ricerca
        self.search_button.clicked.connect(self.search_pypi)

    def search_pypi(self):
        query = self.search_input.text()  # Ottieni il testo inserito dall'utente
        if query:
            self.result_list.clear()  # Cancella i risultati precedenti dalla lista
            url = f"https://pypi.org/simple/{query}/"  # Costruisci l'URL per la ricerca su PyPI
            response = requests.get(url)  # Effettua una richiesta HTTP per ottenere i dati
            if response.status_code == 200:  # Verifica se la richiesta ha avuto successo
                data = response.text  # Ottieni i dati dalla risposta
                package_names = self.extract_package_names(data)  # Estrai i nomi delle librerie
                for package_name in package_names:
                    item = QListWidgetItem(package_name)  # Crea un elemento per la lista
                    self.result_list.addItem(item)  # Aggiungi l'elemento alla lista
```

```
    def extract_package_names(self, data):
        package_names = data.split('\n')  # Suddivide i dati in linee per ottenere i nomi
delle librerie
        return package_names

def main():
    app = QApplication(sys.argv)  # Crea un'applicazione Qt
    window = PyPIBrowser()  # Crea la finestra principale
    window.show()  # Mostra la finestra
    sys.exit(app.exec_())  # Esegui l'applicazione

if __name__ == "__main__":
    main()  # Avvia l'applicazione quando il modulo è eseguito direttamente
```

Analog Clock

Autore: Bocaletto Luca

Sito Web: www.elektronoide.it

Descrizione

Il software "Analog Clock" è un'applicazione per la visualizzazione di un orologio analogico su una finestra grafica. L'orologio visualizza l'orario corrente con lancette per le ore e i minuti, oltre a tacche per i minuti e numeri per le ore.

Caratteristiche Principali

- Visualizzazione dell'orario corrente con lancette delle ore e dei minuti.
- Tacche dei minuti e numeri delle ore per indicare l'orario in modo chiaro.
- Utilizza il modulo PySide6 per la creazione dell'interfaccia grafica.
- Supporta l'antialiasing per ottenere una visualizzazione più fluida.
- Aggiorna l'orologio ogni secondo grazie a un timer.
- Personalizzazione dei colori delle lancette delle ore e dei minuti.

Utilizzo

L'applicazione viene avviata, e un'interfaccia grafica mostra l'orologio analogico. Le lancette delle ore e dei minuti indicano l'orario corrente, mentre le tacche dei minuti e i numeri delle ore facilitano la lettura dell'orario.

Componenti Principali

- **AnalogClockWindow:** Questa classe rappresenta la finestra dell'orologio analogico. Gestisce il rendering dell'orologio e il timer per l'aggiornamento.
- **HOUR_HAND_POLYGON** e **MINUTE_HAND_POLYGON:** Definiscono i poligoni per le lancette delle ore e dei minuti.
- **HOUR_COLOR** e **MINUTE_COLOR:** Specificano i colori per le lancette delle ore e dei minuti.
- **BACKGROUND_COLOR:** Il colore di sfondo dell'orologio.
- **getCurrentTime():** Restituisce l'orario corrente.

Source Code – Analog Clock

```
# Author: Bocaletto Luca
# Web Site: https://www.elektronoide.it
# Software Name: Analog Clock
import sys
```

```
from PySide6.QtCore import QPoint, QTimer, Qt, QTime
from PySide6.QtGui import QColor, QFont, QGradient, QGuiApplication, QPainter,
QPolygon, QRasterWindow
from PySide6.QtWidgets import QApplication
from datetime import datetime

# Definizione delle costanti
HOUR_HAND_POLYGON = QPolygon([QPoint(5, 5), QPoint(-5, 5), QPoint(0, -
50)])  # Lancetta delle ore
MINUTE_HAND_POLYGON = QPolygon([QPoint(3, 3), QPoint(-3, 3), QPoint(0, -
70)])  # Lancetta dei minuti
HOUR_COLOR = QColor(43, 240, 251)  # Colore per la lancetta delle ore
MINUTE_COLOR = QColor(242, 227, 7)  # Colore per la lancetta dei minuti
BACKGROUND_COLOR = QGradient.PlumBath

class AnalogClockWindow(QRasterWindow):

    def __init__(self):
        super().__init__()
        self.setTitle("Analog Clock")
        self.resize(300, 300)  # Dimensioni maggiori

        self._timer = QTimer(self)
        self._timer.timeout.connect(self.update)
        self._timer.start(1000)

    def paintEvent(self, e):
        with QPainter(self) as p:
            self.render(p)

    def render(self, p):
        width = self.width()
        height = self.height()

        p.fillRect(0, 0, width, height, BACKGROUND_COLOR)

        p.setRenderHint(QPainter.Antialiasing)
        p.translate(width / 2, height / 2)

        side = min(width, height)
        p.scale(side / 200.0, side / 200.0)

        current_time = self.getCurrentTime()

        # Disegna le tacche delle decine dei minuti sull'orologio
```

```
        p.setPen(MINUTE_COLOR)
        for i in range(0, 60):
            p.save()
            p.rotate(6.0 * i)
            if i % 5 == 0:
                p.drawLine(0, -70, 0, -80)  # Tacche delle decine dei minuti
            else:
                p.drawLine(0, -75, 0, -80)  # Tacche dei minuti
            p.restore()

        # Disegna i numeri delle ore sull'orologio
        p.setPen(HOUR_COLOR)
        font = QFont("Arial", 14)  # Testo più grande
        p.setFont(font)
        for i in range(1, 13):
            p.save()
            p.rotate(30.0 * i)
            p.translate(0, -90)  # Posiziona i numeri all'esterno dell'orologio con
spaziatura
            p.drawText(-10, 8, str(i))
            p.restore()

        # Disegna la lancetta delle ore
        p.save()
        p.rotate(30.0 * (current_time.hour() + current_time.minute() / 60.0))
        p.setBrush(HOUR_COLOR)
        p.drawConvexPolygon(HOUR_HAND_POLYGON)
        p.restore()

        # Disegna la lancetta dei minuti
        p.save()
        p.rotate(6.0 * (current_time.minute() + current_time.second() / 60.0))
        p.setBrush(MINUTE_COLOR)
        p.drawConvexPolygon(MINUTE_HAND_POLYGON)
        p.restore()

    def getCurrentTime(self):
        return QTime.currentTime()

if __name__ == "__main__":
    app = QGuiApplication(sys.argv)
    clock = AnalogClockWindow()
    clock.show()
    sys.exit(app.exec())
```

Dos Commands Wizard

- **Lingua:** Italiano
- **Autore:** Bocaletto Luca
- **Sito Web:** https://www.elektronoide.it

Descrizione:

Dos Commands Wizard è un'applicazione software scritta in Python che fornisce un'interfaccia grafica per eseguire comandi DOS/Shell in modo interattivo. L'applicazione consente agli utenti di inserire comandi e di eseguirli immediatamente o programmare l'esecuzione in un secondo momento. Inoltre, offre la possibilità di gestire una lista di comandi personalizzati.

Caratteristiche Principali:

- Interfaccia utente grafica basata su Tkinter.
- Esecuzione interattiva di comandi DOS/Shell.
- Possibilità di eseguire comandi immediatamente.
- Programmazione dell'esecuzione di comandi in base a un intervallo di tempo specificato.
- Gestione di comandi personalizzati con nomi e descrizioni.
- Visualizzazione dell'output dei comandi eseguiti con evidenziazione di stdout e stderr.
- Possibilità di annullare la programmazione di comandi in attesa.
- Visualizzazione della directory corrente e possibilità di cambiare la directory.
- Interfaccia utente per la gestione dei comandi personalizzati, inclusi aggiunta, modifica ed eliminazione.

Source Code – Dos Commands Wizard

```
# Software Name: Dos Commands Wizard
# Language: Italian
# Author: Bocaletto Luca
# Web Site: https://www.elektronoide.it
# Importa le librerie necessarie
import tkinter as tk  # Per la creazione dell'interfaccia grafica
import sqlite3  # Per la gestione del database
import subprocess  # Per l'esecuzione dei comandi shell
import threading  # Per la gestione dei thread
import time  # Per il controllo del tempo
import os
# Crea una connessione al database (o lo crea se non esiste)
conn = sqlite3.connect('custom_commands.db')
cursor = conn.cursor()

# Crea la tabella se non esiste
cursor.execute('''CREATE TABLE IF NOT EXISTS commands
```

```
                (id INTEGER PRIMARY KEY,
                 name TEXT,
                 description TEXT)'")

# Definizione della classe principale dell'applicazione
class CommandWindow(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("Dos Commands Wizard")
        self.geometry("600x400")

        # Crea l'interfaccia utente
        self.create_ui()
        self.is_scheduled = False
        self.scheduled_job = None

    def create_ui(self):
        # Creazione degli elementi dell'interfaccia utente
        self.command_entry = tk.Entry(self)
        self.command_entry.pack(fill=tk.X, padx=10, pady=10)

        self.interval_entry = tk.Entry(self)
        self.interval_entry.pack(fill=tk.X, padx=10, pady=5)
        self.interval_entry.insert(0, "5")

        self.schedule_button = tk.Button(self, text="Temporizza Comando",
command=self.schedule_command)
        self.schedule_button.pack(pady=5)

        self.run_now_button = tk.Button(self, text="Esegui Subito",
command=self.run_command_now)
        self.run_now_button.pack(pady=5)

        self.reset_button = tk.Button(self, text="Reset Temporizzazione",
command=self.reset_schedule)
        self.reset_button.pack(pady=5)

        self.custom_commands_button = tk.Button(self, text="Lista Comandi",
command=self.open_custom_commands_window)
        self.custom_commands_button.pack(pady=5)

        self.output_text = tk.Text(self, wrap=tk.WORD)
        self.output_text.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)
        self.output_text.config(state=tk.DISABLED)
```

```
    # Metodo per eseguire un comando
    def run_command(self, command):
        try:
            if command.startswith("cd.."):
                # Esegui il cambio directory
                parent_dir = os.path.abspath(os.path.join(os.getcwd(), os.pardir))
                os.chdir(parent_dir)
                self.show_info(f"Directory corrente: {os.getcwd()}")
            else:
                output = subprocess.check_output(command, shell=True,
stderr=subprocess.STDOUT, text=True)
                self.display_output(output, 'stdout')
        except subprocess.CalledProcessError as e:
            error_output = e.output
            self.display_output(error_output, 'stderr')

    # Metodo per visualizzare l'output dell'esecuzione
    def display_output(self, output, tag):
        self.output_text.config(state=tk.NORMAL)
        self.output_text.insert(tk.END, output, tag)
        self.output_text.tag_add(tag, '1.0', 'end')
        self.output_text.tag_config(tag, foreground='green' if tag == 'stdout' else 'red')
        self.output_text.config(state=tk.DISABLED)

    # Metodo per attivare/disattivare la temporizzazione
    def toggle_scheduling(self):
        self.is_scheduled = not self.is_scheduled

    # Metodo per eseguire un comando in modo programmato
    def schedule_execution(self, command, interval_seconds):
        time.sleep(interval_seconds)
        if self.is_scheduled:
            self.run_command(command)

    # Metodo per visualizzare messaggi informativi
    def show_info(self, message):
        self.output_text.config(state=tk.NORMAL)
        self.output_text.insert(tk.END, message + "\n")
        self.output_text.config(state=tk.DISABLED)

    # Metodo per programmare l'esecuzione di un comando
    def schedule_command(self):
        command = self.command_entry.get()
        interval_str = self.interval_entry.get()
```

```
        if not command:
            self.show_info("Inserisci un comando valido.")
            return

        try:
            interval_seconds = int(interval_str)
        except ValueError:
            self.show_info("Inserisci un intervallo di tempo valido (in secondi).")
            return

        if interval_seconds <= 0:
            self.show_info("L'intervallo di tempo deve essere superiore a 0.")
            return

        self.is_scheduled = True
        self.show_info(f"Temporizzazione attivata: esecuzione del comando
'{command}' tra {interval_seconds} secondo/i...")
        if self.scheduled_job:
            self.scheduled_job.cancel()
        self.scheduled_job = threading.Timer(interval_seconds, self.run_command,
args=(command,))
        self.scheduled_job.start()

    # Metodo per eseguire un comando immediatamente
    def run_command_now(self):
        command = self.command_entry.get()
        if not command:
            self.show_info("Inserisci un comando valido.")
            return
        self.run_command(command)

    # Metodo per annullare la temporizzazione
    def reset_schedule(self):
        self.is_scheduled = False
        if self.scheduled_job:
            self.scheduled_job.cancel()
        self.show_info("Temporizzazione annullata")

    # Metodo per aprire la finestra dei comandi personalizzati
    def open_custom_commands_window(self):
        custom_commands_window = CustomCommandsWindow(self)
        custom_commands_window.title("Lista Comandi Personalizzati")
        custom_commands_window.geometry("600x400")
        custom_commands_window.protocol("WM_DELETE_WINDOW",
self.reload_custom_commands)
```

```
        custom_commands_window.mainloop()

    # Metodo per aggiornare i comandi personalizzati
    def reload_custom_commands(self):
        # Questo metodo viene chiamato quando la finestra "Lista Comandi" viene
chiusa
        # Puoi implementare l'aggiornamento dei comandi personalizzati qui
        pass

# Definizione della classe per la finestra dei comandi personalizzati
class CustomCommandsWindow(tk.Toplevel):
    def __init__(self, parent):
        super().__init__()  # Chiamare il costruttore della superclasse
        self.parent = parent
        self.create_ui()
        self.load_custom_commands()

    def create_ui(self):
        # Creazione dell'interfaccia per i comandi personalizzati
        self.custom_commands_listbox = tk.Listbox(self)
        self.custom_commands_listbox.pack(fill=tk.BOTH, expand=True, padx=10,
pady=10)

        self.name_entry = tk.Entry(self)
        self.name_entry.pack(fill=tk.X, padx=10, pady=5)
        self.name_entry.insert(0, "Nome Comando")

        self.description_entry = tk.Entry(self)
        self.description_entry.pack(fill=tk.X, padx=10, pady=5)
        self.description_entry.insert(0, "Descrizione")

        self.add_button = tk.Button(self, text="Aggiungi Comando",
command=self.add_custom_command)
        self.add_button.pack(pady=5)

        self.edit_button = tk.Button(self, text="Modifica Comando",
command=self.edit_custom_command)
        self.edit_button.pack(pady=5)

        self.delete_button = tk.Button(self, text="Elimina Comando",
command=self.delete_custom_command)
        self.delete_button.pack(pady=5)

        self.custom_commands_listbox.bind("<<ListboxSelect>>",
self.select_custom_command)
```

```
    # Metodo per caricare i comandi personalizzati dal database
    def load_custom_commands(self):
        cursor.execute("SELECT id, name, description FROM commands")
        commands = cursor.fetchall()
        for command in commands:
            self.custom_commands_listbox.insert(tk.END, f"{command[1]} -
{command[2]}")

    # Metodo per selezionare un comando personalizzato
    def select_custom_command(self, event):
        selected_index = self.custom_commands_listbox.curselection()
        if selected_index:
            selected_id = selected_index[0] + 1
            cursor.execute("SELECT name FROM commands WHERE id=?",
(selected_id,))
            result = cursor.fetchone()
            selected_command = result[0]
            self.parent.command_entry.delete(0, tk.END)
            self.parent.command_entry.insert(0, selected_command)  # Inserisci il
comando nella casella di input nella finestra principale

    # Metodo per aggiungere un nuovo comando personalizzato
    def add_custom_command(self):
        name = self.name_entry.get()
        description = self.description_entry.get()
        if name and description:
            cursor.execute("INSERT INTO commands (name, description) VALUES
(?, ?)", (name, description))
            conn.commit()
            self.custom_commands_listbox.insert(tk.END, f"{name} - {description}")

    # Metodo per modificare un comando personalizzato esistente
    def edit_custom_command(self):
        selected_index = self.custom_commands_listbox.curselection()
        if selected_index:
            selected_id = selected_index[0] + 1
            name = self.name_entry.get()
            description = self.description_entry.get()
            if name and description:
                cursor.execute("UPDATE commands SET name=?, description=? WHERE
id=?", (name, description, selected_id))
                conn.commit()
                self.custom_commands_listbox.delete(selected_index)
                self.custom_commands_listbox.insert(tk.END, f"{name} - {description}")
```

```
    # Metodo per eliminare un comando personalizzato
    def delete_custom_command(self):
        selected_index = self.custom_commands_listbox.curselection()
        if selected_index:
            selected_id = selected_index[0] + 1
            cursor.execute("DELETE FROM commands WHERE id=?", (selected_id,))
            conn.commit()
            self.custom_commands_listbox.delete(selected_index)

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

Compressor-Decompressor-Archives

Autore: Bocaletto Luca

Descrizione

Compressor-Decompressor-Archives è un'applicazione desktop sviluppata in Python con l'uso della libreria tkinter per l'interfaccia utente. Questa applicazione consente agli utenti di comprimere e decomprimere file selezionati, supportando i formati di archivio ZIP e GZ. Inoltre, offre la possibilità di visualizzare le informazioni sull'archivio e il suo contenuto.

Funzionalità Principali

- **Comprimere File**: L'applicazione consente agli utenti di selezionare uno o più file e comprimerli in un archivio. Gli utenti possono scegliere tra due formati di archivio supportati: ZIP e GZ.

- **Decomprimere File**: Gli utenti possono selezionare file archiviati nei formati ZIP o GZ e decomprimerli nella directory desiderata.

- **Visualizzare Informazioni sull'Archivio**: L'applicazione permette agli utenti di visualizzare informazioni dettagliate sull'archivio selezionato, inclusi il nome del file, le dimensioni compressa e non compressa, e il metodo di compressione utilizzato. Inoltre, è possibile visualizzare il contenuto dell'archivio.

Utilizzo

1. Avvia l'applicazione e seleziona i file che desideri comprimere, decomprimere o visualizzare.
2. Scegli il formato di archivio desiderato (ZIP o GZ) dall'opzione corrispondente.
3. Utilizza i pulsanti "Comprimi", "Decomprimi" o "Visualizza" per eseguire l'azione desiderata.
4. Le informazioni sullo stato dell'operazione verranno visualizzate nella parte inferiore dell'applicazione.

Source Code – Compressor-Decompressor-Archives

```
# Software Name: Compressor-Decompressor-Archives
# Author: Bocaletto Luca
# Web Site: https://www.elektronoide.it

# Importazione delle librerie necessarie
import tkinter as tk
from tkinter import ttk, filedialog
```

```
import zipfile
import os

# Funzione per comprimere i file selezionati
def compress_files():
    try:
        # Chiede all'utente di selezionare i file da comprimere
        file_paths = filedialog.askopenfilenames()
        if file_paths:
            # Ottiene il tipo di compressione selezionato
            compression_type = compression_var.get()
            output_extension = ".zip" if compression_type == "zip" else ".gz"
            # Chiede all'utente di selezionare la posizione di salvataggio dell'archivio
compresso
            output_path =
filedialog.asksaveasfilename(defaultextension=output_extension)
            if output_path:
                # Crea un oggetto zipfile e aggiunge i file selezionati all'archivio
                with zipfile.ZipFile(output_path, 'w',
compression=zipfile.ZIP_DEFLATED) as zf:
                    for file_path in file_paths:
                        zf.write(file_path, os.path.basename(file_path))  # Usiamo solo il
nome del file
                result_label.config(text="Compressione completata!")
    except Exception as e:
        result_label.config(text=f"Errore durante la compressione: {str(e)}")

# Funzione per decomprimere i file selezionati
def decompress_files():
    try:
        # Chiede all'utente di selezionare i file da decomprimere
        file_paths = filedialog.askopenfilenames()
        if file_paths:
            decompression_type = decompression_var.get()
            for file_path in file_paths:
                if decompression_type == "zip":
                    output_path = file_path + "_decompressed"
                    # Estrae i file dall'archivio ZIP
                    with zipfile.ZipFile(file_path, 'r') as zf:
                        zf.extractall(output_path)
                elif decompression_type == "gz":
                    output_path = file_path[:-3]  # Rimuovi l'estensione .gz
                    with open(output_path, 'wb') as output_file, open(file_path, 'rb') as
input_file:
                        import zlib
```

```
                    decompressor = zlib.decompressobj(16 + zlib.MAX_WBITS)
                    for data in iter(lambda: input_file.read(1024), b''):
                        output_file.write(decompressor.decompress(data))
            result_label.config(text="Decompressione completata!")
    except Exception as e:
        result_label.config(text=f"Errore durante la decompressione: {str(e)}")

# Funzione per visualizzare le informazioni dell'archivio e il suo contenuto
def view_files():
    try:
        # Chiede all'utente di selezionare l'archivio da visualizzare
        file_paths = filedialog.askopenfilenames()
        if file_paths:
            # Abilita il widget di testo per l'output
            output_text.config(state=tk.NORMAL)
            output_text.delete(1.0, tk.END)
            archive_files = []

            for file_path in file_paths:
                output_text.insert(tk.END, f"Informazioni su {file_path}:\n")
                with zipfile.ZipFile(file_path, 'r') as zf:
                    for info in zf.infolist():
                        output_text.insert(tk.END, f"Nome: {info.filename}\n")
                        output_text.insert(tk.END, f"Dimensione compressa:
{info.compress_size} bytes\n")
                        output_text.insert(tk.END, f"Dimensione non compressa:
{info.file_size} bytes\n")
                        output_text.insert(tk.END, f"Metodo di compressione:
{info.compress_type}\n")
                        output_text.insert(tk.END, "\n")
                        archive_files.extend(zf.namelist())

            output_text.insert(tk.END, "Contenuto dell'Archivio:\n")
            for file_name in archive_files:
                output_text.insert(tk.END, file_name + "\n")

            # Disabilita il widget di testo per l'output
            output_text.config(state=tk.DISABLED)
    except Exception as e:
        result_label.config(text=f"Errore durante la visualizzazione: {str(e)}")

# Creazione della finestra principale
app = tk.Tk()
app.title("Software di Compressione/Decompressione")
```

```
# Creazione del frame principale
file_frame = tk.Frame(app, padx=10, pady=10)
file_frame.grid(row=0, column=0, columnspan=4)

# Etichette e opzioni per la compressione e la decompressione
title_label = tk.Label(file_frame, text="Seleziona i file da comprimere, decomprimere
o visualizzare:")
title_label.grid(row=0, column=0, columnspan=4)

compression_label = tk.Label(file_frame, text="Seleziona il formato di archivio:")
compression_label.grid(row=1, column=0)
compression_var = tk.StringVar()
compression_var.set("zip")
compression_options = ["zip", "gz"]
compression_menu = tk.OptionMenu(file_frame, compression_var,
*compression_options)
compression_menu.grid(row=1, column=1)

decompression_label = tk.Label(file_frame, text="Seleziona il formato di archivio:")
decompression_label.grid(row=2, column=0)
decompression_var = tk.StringVar()
decompression_var.set("zip")
decompression_options = ["zip", "gz"]
decompression_menu = tk.OptionMenu(file_frame, decompression_var,
*decompression_options)
decompression_menu.grid(row=2, column=1)

# Pulsanti per le azioni di compressione, decompressione e visualizzazione
compress_button = tk.Button(file_frame, text="Comprimi",
command=compress_files)
compress_button.grid(row=1, column=2)

decompress_button = tk.Button(file_frame, text="Decomprimi",
command=decompress_files)
decompress_button.grid(row=2, column=2)

view_button = tk.Button(file_frame, text="Visualizza", command=view_files)
view_button.grid(row=3, column=1)

# Barra di avanzamento
progress = ttk.Progressbar(app, length=300, mode='determinate')
progress.grid(row=1, column=0, columnspan=4, pady=10)

# Widget di testo per l'output
output_text = tk.Text(app, wrap=tk.WORD, width=40, height=10)
```

```
output_text.grid(row=2, column=0, columnspan=4, pady=10)
output_text.config(state=tk.DISABLED)

# Etichetta per i risultati
result_label = tk.Label(app, text="")
result_label.grid(row=3, column=0, columnspan=4)

# Esecuzione dell'app
app.mainloop()
```

Creatore di USB Avviabile

Il Creatore di USB Avviabile è uno script Python che fornisce un'interfaccia utente grafica semplice per creare unità USB avviabili da file ISO. Questo strumento è stato creato da Luca Bocaletto.

Funzionalità

- **Seleziona File ISO**: Scegli un file ISO per creare un'unità USB avviabile.
- **Rilevamento Automatico USB**: Rileva automaticamente le unità USB disponibili.
- **Convalida dell'Unità**: Verifica se il dispositivo selezionato è un'unità USB valida.
- **Formattazione dell'Unità USB**: Formatta l'unità USB con il sistema di file FAT32.
- **Copia del File ISO**: Copia il contenuto del file ISO selezionato sull'unità USB.

Requisiti

Per utilizzare il Creatore di USB Avviabile, sono necessari i seguenti requisiti:

- Python 3 installato nel sistema.
- Pacchetti Python necessari: `tkinter`, `win32file`, `win32com.client`.

Utilizzo

1. Esegui lo script utilizzando Python 3.
2. Seleziona il file ISO facendo clic sul pulsante "Sfoglia".
3. Scegli un'unità USB dalla lista delle unità disponibili.
4. Fai clic sul pulsante "Crea Unità USB Avviabile" per avviare il processo.

Come Funziona

Il Creatore di USB Avviabile utilizza la Windows Management Instrumentation (WMI) e le API Win32 per identificare le unità USB e gestire il processo di creazione di un'unità USB avviabile.

Ecco come funziona:

- Lo script utilizza la WMI per identificare le unità USB verificando il campo `InterfaceType`.
- Convalida il percorso dell'unità USB selezionata e si assicura che sia un dispositivo rimovibile.
- Lo script formatta l'unità USB utilizzando il sistema di file FAT32.
- Successivamente, copia il contenuto del file ISO selezionato sull'unità USB utilizzando il comando `xcopy`.

Si prega di notare che è necessario disporre dei privilegi necessari per formattare l'unità USB.

Source Code – Creatore di USB Avviabile

```
# Software Name: Bootable USB
# Author: Luca Bocaletto
# Website: https://www.elektronoide.it

import tkinter as tk
from tkinter import filedialog
import subprocess
import os
import win32file
import win32com.client
import tkinter.ttk as ttk

# Funzione per prendere la USB
def get_usb_drives():
    drives = []
    wmi = win32com.client.GetObject("winmgmts:")
    for disk in wmi.InstancesOf("Win32_DiskDrive"):
        if "USB" in disk.InterfaceType:
            partitions = disk.Associators_("Win32_DiskDriveToDiskPartition")
            for partition in partitions:
                logical_disks = partition.Associators_("Win32_LogicalDiskToPartition")
                for logical_disk in logical_disks:
                    drives.append(logical_disk.DeviceID)
    return drives

# Funzione per la selezione del file ISO
def select_iso():
    file_path = filedialog.askopenfilename(filetypes=[("ISO files", "*.iso")])
    if file_path:
        iso_entry.delete(0, tk.END)
        iso_entry.insert(0, file_path)

# Funzione per crea la USB Bootable
def create_bootable_usb():
    iso_file = iso_entry.get()
    selected_drive = usb_combobox.get()

    if not iso_file:
        result_label.config(text="Per favore, seleziona il file ISO.")
        return

    if not selected_drive:
```

```
        result_label.config(text="Per favore, seleziona una chiavetta USB.")
        return

    try:
        # Verifica se il percorso della chiavetta USB è valido
        if not os.path.exists(selected_drive):
            raise Exception("Il percorso della chiavetta USB selezionata non è valido.")

        # Verifica se il dispositivo selezionato è una chiavetta USB
        drive_type = win32file.GetDriveType(selected_drive)
        if drive_type != win32file.DRIVE_REMOVABLE:
            raise Exception("Il dispositivo selezionato non è una chiavetta USB.")

        # Formattazione della chiavetta USB (assicurati di avere i privilegi per farlo)
        format_command = f"format {selected_drive} /FS:FAT32 /Q"
        subprocess.run(format_command, shell=True, check=True)

        # Scrittura dell'ISO sulla chiavetta USB
        write_command = f"xcopy {iso_file} {selected_drive} /s /e /f"
        subprocess.run(write_command, shell=True, check=True)

        result_label.config(text="Il supporto USB avviabile è stato creato con
successo!")
    except Exception as e:
        result_label.config(text=f"Errore: {str(e)}")

# Creazione della finestra principale
window = tk.Tk()
window.title("Crea Supporto USB Avviabile")

# Spiegazione dell'utilizzo
usage_label = tk.Label(window, text="Utilizzo:\n\n"
                               "1. Seleziona il file ISO utilizzando il pulsante 'Sfoglia'.\n"
                               "2. Scegli una chiavetta USB dalla lista.\n"
                               "3. Fai clic su 'Crea Supporto USB Avviabile' per avviare il
processo.")
usage_label.pack()

# Etichetta e campo di inserimento per il percorso ISO
iso_label = tk.Label(window, text="Seleziona il file ISO:")
iso_label.pack()
iso_entry = tk.Entry(window)
iso_entry.pack()
browse_button = tk.Button(window, text="Sfoglia", command=select_iso)
browse_button.pack()
```

```
# Etichetta e ComboBox per la selezione del dispositivo USB
usb_label = tk.Label(window, text="Seleziona una chiavetta USB:")
usb_label.pack()
usb_drives = get_usb_drives()
usb_combobox = ttk.Combobox(window, values=usb_drives)
usb_combobox.pack()

# Pulsante per avviare il processo di creazione del supporto USB avviabile
create_button = tk.Button(window, text="Crea Supporto USB Avviabile",
command=create_bootable_usb)
create_button.pack()

# Barra di avanzamento
progress = ttk.Progressbar(window, length=200, mode="indeterminate")
progress.pack()

# Etichetta per visualizzare i risultati
result_label = tk.Label(window, text="")
result_label.pack()

# Esegui il ciclo principale della GUI
window.mainloop()
```

Browser Web

Un'applicazione semplice di browser web realizzata utilizzando Python e PyQt5.

Funzionalità

- Naviga su siti web inserendo URL o query di ricerca.
- Navigazione avanti e indietro.
- Ricarica la pagina corrente.
- Imposta DuckDuckGo come pagina iniziale.
- Impostazioni avanzate di sicurezza e privacy per disabilitare JavaScript e bloccare le risorse di terze parti.

Installazione

1. Clona il repository:

   ```
   git clone https://github.com/elektronoide/browser-web.git
   ```

Utilizzo

- **Inserisci un URL del sito web o una query di ricerca:** Nella barra degli indirizzi, puoi inserire l'URL di un sito web o una query di ricerca e premere Invio per navigare alla pagina desiderata.

- **Pulsanti di navigazione:** Usa i pulsanti indietro e avanti per muoverti nella cronologia di navigazione e rivedere le pagine visitate in precedenza.

- **Pulsante di ricarica:** Clicca sul pulsante di ricarica per aggiornare la pagina corrente e ottenere i contenuti più recenti.

- **Pulsante Home:** Cliccando sul pulsante Home, tornerai alla pagina iniziale, impostata come DuckDuckGo per impostazione predefinita.

Source Code – Broser Web

```
# Software Name: Browser Web
# Author: Bocaletto Luca
# Site Web: https://www.elektronoide.it

import sys
from PyQt5.QtCore import *
from PyQt5.QtWidgets import *
from PyQt5.QtWebEngineWidgets import *
```

```
class WebBrowser(QMainWindow):
    def __init__(self):
        super().__init__()

        self.browser = QWebEngineView()
        self.url_bar = QLineEdit()
        self.url_bar.returnPressed.connect(self.navigate)

        nav_bar = QToolBar()
        nav_bar.addWidget(self.url_bar)
        self.addToolBar(nav_bar)

        # Pulsante per tornare indietro
        back_btn = QAction("Indietro", self)
        back_btn.setStatusTip("Torna alla pagina precedente")
        back_btn.triggered.connect(self.browser.back)
        nav_bar.addAction(back_btn)

        # Pulsante per avanzare
        forward_btn = QAction("Avanti", self)
        forward_btn.setStatusTip("Vai alla pagina successiva")
        forward_btn.triggered.connect(self.browser.forward)
        nav_bar.addAction(forward_btn)

        # Pulsante per ricaricare la pagina
        reload_btn = QAction("Ricarica", self)
        reload_btn.setStatusTip("Ricarica la pagina corrente")
        reload_btn.triggered.connect(self.browser.reload)
        nav_bar.addAction(reload_btn)

        # Pulsante per tornare alla pagina iniziale
        home_btn = QAction("Home", self)
        home_btn.setStatusTip("Torna alla pagina iniziale")
        home_btn.triggered.connect(self.navigate_home)
        nav_bar.addAction(home_btn)

        # Aggiunge un'azione per aggiornare l'URL nella barra degli indirizzi quando
cambia la pagina
        self.browser.urlChanged.connect(self.update_urlbar)

        self.setCentralWidget(self.browser)
        self.setWindowTitle("Web Browser")
        self.setGeometry(100, 100, 1024, 768)  # Imposta la dimensione minima

        # Impostazioni avanzate di sicurezza e privacy
```

```
        self.setup_security_settings()

    def navigate_home(self):
        self.browser.setUrl(QUrl("https://duckduckgo.com"))  # Imposta DuckDuckGo
come pagina iniziale

    def navigate(self):
        user_input = self.url_bar.text()
        if "." in user_input:
            url = QUrl.fromUserInput(user_input)
        else:
            search_query = user_input
            url = QUrl("https://duckduckgo.com/?q={}".format(search_query))

        self.browser.setUrl(url)

    def update_urlbar(self, q):
        self.url_bar.setText(q.toString())
        self.url_bar.setCursorPosition(0)

    def setup_security_settings(self):
        # Disabilita l'esecuzione di JavaScript (potenziale fonte di tracciamento)

self.browser.page().settings().setAttribute(QWebEngineSettings.JavascriptEnabled,
False)

        # Blocco delle richieste di risorse di terze parti (possibili tracker)
        self.browser.page().profile().cookieStore().deleteAllCookies()  # Cancella i
cookie

    def block_third_party_requests(self, info):
        if info.requestUrl().host() != info.firstPartyRequestUrl().host():
            info.block(True)

if __name__ == '__main__':
    app = QApplication(sys.argv)
    app.setApplicationName("Web Browser")
    window = WebBrowser()
    window.show()
    app.exec_()
```

Desktop Recorder

Desktop Recorder è un'applicazione desktop in Python che ti consente di registrare il tuo schermo in vari formati di qualità video. Puoi selezionare tra diverse opzioni di qualità, come 480p, 720p e 1080p, per le tue registrazioni.

Caratteristiche

- Registra il tuo schermo in vari formati di qualità video.
- Opzioni di qualità video, tra cui 480p, 720p e 1080p.
- Interfaccia utente semplice per iniziare e arrestare le registrazioni.
- Visualizza una miniatura in tempo reale della registrazione.
- Notifiche per l'inizio e la fine della registrazione.

Source Code – Desktop Recorder

```
# Software Name: Desktop Recorder
# Author: Bocaletto Luca
# Site Web: https://www.elektronoide.it
# Language: Italian
import sys
import cv2
import pyautogui
import numpy as np
from PyQt5.QtWidgets import QApplication, QMainWindow, QPushButton,
QVBoxLayout, QWidget, QRadioButton, QLabel
from PyQt5.QtCore import QTimer
from plyer import notification
from PyQt5.QtGui import QImage, QPixmap

# Definizione della classe principale
class DesktopRecorder(QMainWindow):
    def __init__(self):
        super().__init__()

        self.initUI()

        self.is_recording = False
        self.video_writer = None
        self.selected_quality = None

    # Inizializzazione dell'interfaccia utente
    def initUI(self):
        self.setWindowTitle("Desktop Recorder")
```

```
        self.setGeometry(100, 100, 800, 600)

        self.central_widget = QWidget()
        self.setCentralWidget(self.central_widget)

        self.layout = QVBoxLayout()

                        title_label = QLabel("Screen Recorder", self)  # Aggiungi un
titolo
        title_label.setAlignment(Qt.AlignCenter)  # Imposta l'allineamento al centro
        self.layout.addWidget(title_label)

        # Bottone per iniziare/arrestare la registrazione
        self.record_btn = QPushButton("Inizia a Registrare")
        self.record_btn.clicked.connect(self.toggle_recording)
        self.layout.addWidget(self.record_btn)

        # Opzioni di qualità video
        self.low_quality_radio = QRadioButton("480p")
        self.medium_quality_radio = QRadioButton("720p")
        self.high_quality_radio = QRadioButton("1080p")
        self.quality_group = []  # Gruppo per gestire le opzioni di qualità

        self.low_quality_radio.setChecked(True)  # Opzione predefinita
        self.layout.addWidget(self.low_quality_radio)
        self.layout.addWidget(self.medium_quality_radio)
        self.layout.addWidget(self.high_quality_radio)

        # Configura la finestra principale con il layout
        self.central_widget.setLayout(self.layout)

        # Timer per catturare il frame
        self.timer = QTimer(self)
        self.timer.timeout.connect(self.update_frame)
        self.timer.start(20)  # Aggiorna il frame ogni 20 ms

        # Label per visualizzare la miniatura del video
        self.thumbnail_label = QLabel(self)
        self.thumbnail_label.setFixedSize(300, 420)
        self.layout.addWidget(self.thumbnail_label)

    # Funzione per iniziare/arrestare la registrazione
    def toggle_recording(self):
        if not self.is_recording:
            for radio in self.quality_group:
```

```
            if radio.isChecked():
                self.selected_quality = radio.text()
                break

        # Configurazione del video writer
        fourcc = cv2.VideoWriter_fourcc(*'XVID')
        screen_width, screen_height = pyautogui.size()
        output_filename = f"desktop_recording_{self.selected_quality}.avi"

        self.video_writer = cv2.VideoWriter(output_filename, fourcc, 20.0,
(screen_width, screen_height))
        self.record_btn.setText("Arresta la Registrazione")
        self.is_recording = True

        # Notifica l'inizio della registrazione
        notification.notify(
            title="Desktop Recorder",
            message="La registrazione è iniziata.",
        )
    else:
        # Arresta la registrazione e rilascia il writer
        self.video_writer.release()
        self.record_btn.setText("Inizia a Registrare")
        self.is_recording = False

        # Notifica la fine della registrazione
        notification.notify(
            title="Desktop Recorder",
            message="La registrazione è stata interrotta.",
        )

# Funzione per catturare e aggiornare il frame
def update_frame(self):
    if self.is_recording:
        screenshot = pyautogui.screenshot()
        frame = cv2.cvtColor(np.array(screenshot), cv2.COLOR_RGB2BGR)
        self.video_writer.write(frame)

        small_frame = cv2.resize(frame, (300, 420))
        frame_rgb = cv2.cvtColor(small_frame, cv2.COLOR_BGR2RGB)
        h, w, ch = frame_rgb.shape
        bytes_per_line = ch * w
        q_image = QImage(frame_rgb.data, w, h, bytes_per_line,
QImage.Format_RGB888)
        self.thumbnail_label.setPixmap(QPixmap.fromImage(q_image))
```

```
# Funzione principale
def main():
    app = QApplication(sys.argv)
    window = DesktopRecorder()
    window.show()
    sys.exit(app.exec_())

if __name__ == "__main__":
    main()
```

Info PC/OS/NET

Descrizione

Info PC/OS/NET è un'applicazione Python che raccoglie e visualizza informazioni sul tuo computer, sistema operativo e rete. Fornisce dettagli sul tuo hardware, software e configurazione di rete in un'interfaccia grafica amichevole.

Caratteristiche

- Visualizza informazioni dettagliate sul tuo computer, inclusi CPU, GPU, RAM e disco rigido.
- Mostra informazioni di rete, tra cui indirizzo IP e indirizzo MAC.
- Ottieni informazioni sul sistema operativo, tra cui la versione e le impostazioni di lingua.
- Ottieni informazioni sul tuo indirizzo IP e provider.
- Interfaccia utente amichevole utilizzando Tkinter.

Source Code – Info PC

```
# Software Name: Info PC/OS/Net
# Author: Bocaletto Luca
# Site Web: https://www.elektronoide.it
# License: GPLv3

import tkinter as tk
import platform
import psutil
import socket
import GPUtil
import requests
import uuid
import wmi
```

```
import getpass
import locale
import time

# Funzione per ottenere informazioni sulla scheda madre
def get_motherboard_info():
    c = wmi.WMI()
    motherboard_info = [
        ("Produttore Scheda Madre", c.Win32_BaseBoard()[0].Manufacturer),
        ("Chipset Scheda Madre", c.Win32_BaseBoard()[0].Product)
    ]
    return motherboard_info

# Funzione per ottenere informazioni sul sistema operativo e il computer
def get_system_info():
    system_info = [
        ("Sistema Operativo", f"{platform.system()} {platform.release()}"),
        ("Versione di Build", platform.win32_ver()[1]),
        ("Nome Utente", getpass.getuser()),  # Aggiunto Nome Utente
        ("Architettura", platform.architecture()),  # Aggiunta Architettura
        ("Lingua del Sistema", locale.getlocale()[0]),  # Aggiunta Lingua del Sistema
        ("Fuso Orario", time.tzname[0])  # Aggiunto Fuso Orario
    ]
    return system_info


# Funzione per ottenere l'indirizzo IP con il formato desiderato
def get_ip_address():
    hostname = socket.gethostname()
    ip_address = socket.gethostbyname(hostname)
    return [("Indirizzo IP", ip_address)]

# Funzione per ottenere il provider IP
def get_ip_provider():
    try:
        response = requests.get("https://ipinfo.io/json")
        data = response.json()
        ip = data.get("ip", "IP sconosciuto")
        provider = data.get("org", "Provider IP sconosciuto")
        return [("Indirizzo IP", ip), ("Provider IP", provider)]
    except Exception as e:
        return [("Indirizzo IP", "N/A"), ("Provider IP", "N/A")]

def get_network_info():
    try:
```

```
        hostname = socket.gethostname()
        ip_address = socket.gethostbyname(hostname)
        mac_address = ':'.join(['{:02x}'.format((uuid.getnode() >> elements) & 0xff) for
elements in range(5, -1, -1)])
        network_info = [
            ("Indirizzo IP", ip_address),
            ("Indirizzo MAC", mac_address)
        ]
        return network_info
    except Exception as e:
        return [("Informazioni sulla rete", "N/A")]

# Funzione per ottenere informazioni sulla CPU
def get_cpu_info():
    cpu_info = [
        ("CPU", f"{psutil.cpu_percent()}% in uso"),
        ("Nome Processore", platform.processor()),
        ("Numero di Core", psutil.cpu_count(logical=False)),
        ("Numero di Thread", psutil.cpu_count(logical=True))
    ]
    return cpu_info

# Funzione per ottenere informazioni sulla GPU, inclusa la VRAM in GB
def get_gpu_info():
    try:
        gpu_info = GPUtil.getGPUs()[0]
        gpu_name = gpu_info.name
        gpu_load = gpu_info.load
        gpu_memory_total_mb = gpu_info.memoryTotal
        gpu_memory_free_mb = gpu_info.memoryFree
        gpu_memory_total_gb = gpu_memory_total_mb / 1024  # Conversione da MB a
GB
        gpu_memory_free_gb = gpu_memory_free_mb / 1024  # Conversione da MB a
GB
        return [
            ("GPU", gpu_name),
            ("Utilizzo GPU", f"{gpu_load * 100:.2f}%"),
            ("VRAM totale", f"{gpu_memory_total_gb:.2f} GB"),
            ("VRAM libera", f"{gpu_memory_free_gb:.2f} GB")
        ]
    except Exception as e:
        return [("GPU", "N/A")]

# Funzione per ottenere informazioni sulle risorse del sistema
def get_system_resources():
```

```
    ram_info = [
        ("RAM totale", f"{psutil.virtual_memory().total/1e9:.2f} GB"),
        ("RAM in uso", f"{psutil.virtual_memory().used/1e9:.2f} GB")
    ]

    def get_hdd_info():
        partitions = psutil.disk_partitions()
        hdd_info = [("HDD", "")]
        for partition in partitions:
            hdd_info.append(
                (f"{partition.device}", f"{psutil.disk_usage(partition.device).total/1e9:.2f}
GB totale, {psutil.disk_usage(partition.device).used/1e9:.2f} GB in uso")
            )
        return hdd_info

    hdd_info = get_hdd_info()
    ip_provider = get_ip_provider()
    network_info = get_network_info()

    return get_motherboard_info() + get_cpu_info() + ram_info + get_gpu_info() +
hdd_info + ip_provider + network_info

# Funzione per ottenere tutte le informazioni
def get_all_info():
    info = get_system_info() + get_system_resources()
    return info

# Creazione della finestra principale
root = tk.Tk()
root.title("INFO PC/OS/NET")

# Creazione di una tabella
for i, (name, data) in enumerate(get_all_info()):
    label_name = tk.Label(root, text=name, font=("Arial", 12))
    label_name.grid(row=i, column=0, sticky="W")

    if isinstance(data, list):  # Verifica se i dati sono una lista (per l'HDD, GPU e altri)
        for j, (sub_name, sub_data) in enumerate(data):
            label_sub_name = tk.Label(root, text=sub_name, font=("Arial", 12))
            label_sub_name.grid(row=i + j, column=1, sticky="W")

            label_sub_data = tk.Label(root, text=sub_data, font=("Arial", 12))
            label_sub_data.grid(row=i + j, column=2, sticky="W")
    else:
        label_data = tk.Label(root, text=data, font=("Arial", 12))
```

```
    label_data.grid(row=i, column=1, columnspan=2, sticky="W")

# Esecuzione del loop principale della GUI
root.mainloop()
```

Translate

Autore: Bocaletto Luca
Licenza: GPLv3

Panoramica

Translate è un'applicazione Python che semplifica il processo di traduzione di testo da una lingua all'altra utilizzando l'API di Google Translate. Questo strumento open source fornisce un'interfaccia utente grafica facile da usare per consentire agli utenti di inserire testo, selezionare lingue di origine e destinazione e ottenere traduzioni in modo rapido.

Caratteristiche

- Traduzione di testo: Traduci facilmente il testo da una lingua all'altra.
- Interfaccia utente intuitiva: GUI intuitiva per la traduzione del testo senza problemi.
- Opzioni di lingua: Supporta una varietà di lingue di origine e destinazione.
- Gestione degli errori: Fornisce messaggi di errore informativi in caso di problemi di traduzione.

Per iniziare

Uso

1. Inserisci il testo che desideri tradurre nella casella di testo di input.
2. Seleziona la lingua di origine dal menu a discesa "Lingua di origine".
3. Scegli la lingua di destinazione dal menu a discesa "Lingua di destinazione".
4. Fai clic sul pulsante "Traduci".
5. Il testo tradotto apparirà nella casella di testo di output.

```
# Name Software: Translate GUI Italian
# Author: Bocaletto Luca
# License: GPLv3
import tkinter as tk
from tkinter import ttk
```

```
from googletrans import Translator

def translate_text():
    # Ottieni il testo da tradurre dalla casella di testo di input
    text_to_translate = input_text.get("1.0", "end-1c")
    translator = Translator()

    # Ottieni la lingua di origine e la lingua di destinazione selezionate dall'utente
    src_lang = source_language.get()
    dest_lang = destination_language.get()

    try:
        # Utilizza il traduttore di Google per tradurre il testo
        translated_result = translator.translate(text_to_translate, src=src_lang,
dest=dest_lang)
        if translated_result.text is not None:
            # Se la traduzione ha successo, mostra il testo tradotto nell'area di output
            translated_text = translated_result.text
            output_text.config(state="normal")
            output_text.delete("1.0", "end")  # Cancella il campo di output
            output_text.insert("1.0", translated_text)  # Inserisci il testo tradotto
            output_text.config(state="disabled")
        else:
            # Se la traduzione non ha successo, mostra un messaggio di errore nell'area di
output
            output_text.config(state="normal")
            output_text.delete("1.0", "end")
            output_text.insert("1.0", "La traduzione non è riuscita. Riprova.")
            output_text.config(state="disabled")
    except Exception as e:
        # Gestisci eventuali eccezioni mostrando un messaggio di errore nell'area di
output
        output_text.config(state="normal")
        output_text.delete("1.0", "end")
        output_text.insert("1.0", "Errore nella traduzione: " + str(e))
        output_text.config(state="disabled")

# Crea la finestra principale
root = tk.Tk()
root.title("Traduttore")

# Crea una casella di testo per l'input
input_text = tk.Text(root, wrap=tk.WORD, width=40, height=10)
input_text.grid(row=0, column=0, padx=10, pady=10)
```

```
# Crea una casella di testo per l'output (disabilitata inizialmente)
output_text = tk.Text(root, wrap=tk.WORD, width=40, height=10, state="disabled")
output_text.grid(row=0, column=1, padx=10, pady=10)

# Crea le etichette per le lingue
source_language_label = ttk.Label(root, text="Lingua di origine:")
source_language_label.grid(row=1, column=0, padx=10, pady=5, sticky="w")
destination_language_label = ttk.Label(root, text="Lingua di destinazione:")
destination_language_label.grid(row=2, column=0, padx=10, pady=5, sticky="w")

# Crea le opzioni per le lingue
languages = ["en", "es", "fr", "de", "it", "ja", "ko", "zh-CN", "ru"]
source_language = ttk.Combobox(root, values=languages)
source_language.grid(row=1, column=1, padx=10, pady=5)
source_language.set("en")  # Imposta la lingua di origine predefinita
destination_language = ttk.Combobox(root, values=languages)
destination_language.grid(row=2, column=1, padx=10, pady=5)
destination_language.set("it")  # Imposta la lingua di destinazione predefinita

# Crea il pulsante di traduzione
translate_button = ttk.Button(root, text="Traduci", command=translate_text)
translate_button.grid(row=3, columnspan=2, padx=10, pady=10)

# Esegui il ciclo principale della GUI
root.mainloop()
```

Input PC Recorder

Autore: Bocaletto Luca
Sito Web: elektronoide.it

Panoramica

Input PC Recorder è una semplice applicazione Python per registrare l'audio dai dispositivi di input del tuo PC. Fornisce un'interfaccia utente intuitiva per selezionare dispositivi di input, profondità in bit e frequenza di campionamento, e consente di avviare e interrompere le registrazioni audio.

Caratteristiche

- Seleziona il dispositivo audio di input.
- Scegli la profondità in bit (8-bit, 16-bit, 24-bit) e la frequenza di campionamento (44100 Hz, 48000 Hz, 96000 Hz, 192000 Hz).
- Avvia e interrompi la registrazione audio.

- Monitora l'intensità della registrazione in tempo reale.
- Salva l'audio registrato in un file WAV.

Uso

1. Avvia l'applicazione.
2. Seleziona il dispositivo di input desiderato dal menu a tendina.
3. Scegli la profondità in bit e la frequenza di campionamento.
4. Fai clic sul pulsante "Avvia Registrazione" per iniziare la registrazione audio.
5. Fai clic sul pulsante "Interrompi Registrazione" per interrompere la registrazione e salvare l'audio in un file WAV.
6. L'intensità della registrazione viene visualizzata in tempo reale.

```
# Software Name: Input PC Recorder
# Author: Bocaletto Luca
# Site Web: https://www.elektronoide.it
# License: GPLv3
import sys
import numpy as np
import sounddevice as sd
import soundfile as sf
from PyQt5.QtWidgets import QApplication, QMainWindow, QVBoxLayout,
QWidget, QComboBox, QPushButton, QLabel, QMessageBox
from PyQt5.QtCore import QTimer

# Definizione della classe principale dell'app
class AudioRecorderApp(QMainWindow):
    def __init__(self):
        super().__init__()

        # Impostazioni della finestra principale
        self.setWindowTitle("Audio Recorder")  # Imposta il titolo della finestra
principale
        self.setGeometry(100, 100, 400, 200)  # Imposta posizione e dimensioni della
finestra

        # Creazione del widget centrale
        self.central_widget = QWidget()
        self.setCentralWidget(self.central_widget)

        # Creazione del layout principale
        self.layout = QVBoxLayout()

        # Creazione dei widget dell'interfaccia utente
        self.device_combo = QComboBox()  # Menù a discesa per la selezione del
```

```
dispositivo
        self.layout.addWidget(self.device_combo)

        self.bit_depth_combo = QComboBox()  # Menù a discesa per la profondità in
bit
        self.bit_depth_combo.addItems(["8 bit", "16 bit", "24 bit"])
        self.layout.addWidget(self.bit_depth_combo)

        self.sample_rate_combo = QComboBox()  # Menù a discesa per la frequenza di
campionamento
        self.sample_rate_combo.addItems(["44100 Hz", "48000 Hz", "96000 Hz",
"192000 Hz"])
        self.layout.addWidget(self.sample_rate_combo)

        self.start_button = QPushButton("Avvia Registrazione")  # Pulsante per avviare
la registrazione
        self.stop_button = QPushButton("Interrompi Registrazione")  # Pulsante per
interrompere la registrazione
        self.layout.addWidget(self.start_button)
        self.layout.addWidget(self.stop_button)

        self.intensity_label = QLabel("Intensità: 0")  # Etichetta per mostrare l'intensità
        self.layout.addWidget(self.intensity_label)

        self.central_widget.setLayout(self.layout)

        # Variabili di stato e dati dell'audio
        self.is_recording = False
        self.selected_device_index = None
        self.audio_data = []

        # Connetti i pulsanti agli slot
        self.start_button.clicked.connect(self.start_recording)
        self.stop_button.clicked.connect(self.stop_recording)

        # Imposta un timer per l'aggiornamento dell'intensità
        self.timer = QTimer()
        self.timer.timeout.connect(self.update_intensity)
        self.timer.start(1000)

        # Ottieni la lista dei dispositivi audio
        self.device_info = sd.query_devices()
        self.device_names = []  # Lista per memorizzare i nomi dei dispositivi senza
duplicati
```

```
        for i, info in enumerate(self.device_info):
            if info["max_input_channels"] > 0:
                device_name = f"{info['name']} ({info['max_input_channels']} canali)"
                if device_name not in self.device_names:
                    self.device_names.append(device_name)
                    self.device_combo.addItem(device_name)

        self.p = None
        self.sample_rate = 44100
        self.bit_depth = 16

    def start_recording(self):
        if self.is_recording:
            return

        self.selected_device_index = self.device_combo.currentIndex()
        if self.selected_device_index < 0:
            QMessageBox.critical(self, "Errore", "Seleziona un dispositivo di input
valido.")
            return

        bit_depth_text = self.bit_depth_combo.currentText()
        sample_rate_text = self.sample_rate_combo.currentText()

        self.sample_rate = int(sample_rate_text.split()[0])
        self.bit_depth = int(bit_depth_text.split()[0])

        try:
            if self.device_info[self.selected_device_index]["max_input_channels"] < 1:
                QMessageBox.critical(self, "Errore", "Il dispositivo selezionato non
supporta l'input audio.")
                return

            self.is_recording = True
            self.audio_data = []

            self.p = sd.InputStream(device=self.selected_device_index, channels=1,
samplerate=self.sample_rate, callback=self.audio_callback)
            self.p.start()
            QMessageBox.information(self, "Registrazione Avviata", "La registrazione è
iniziata.")
        except Exception as e:
            QMessageBox.critical(self, "Errore", f"Errore durante l'avvio della
registrazione: {str(e)}")
```

```
    def stop_recording(self):
        if not self.is_recording:
            return

        self.is_recording = False
        if self.p:
            self.p.stop()
            self.p.close()

            if self.audio_data:
                try:
                    self.save_audio_file()
                    QMessageBox.information(self, "Registrazione Interrotta", "La
registrazione è stata interrotta e il file audio è stato salvato.")
                except Exception as e:
                    QMessageBox.critical(self, "Errore", f"Errore durante il salvataggio del
file audio: {str(e)}")

    def audio_callback(self, in_data, frames, time, status):
        if self.is_recording:
            audio_chunk = np.frombuffer(in_data, dtype=np.int16)
            self.audio_data.extend(audio_chunk)

    def update_intensity(self):
        if self.is_recording:
            if len(self.audio_data) > 0:
                intensity = int(np.abs(np.mean(self.audio_data)))
                self.intensity_label.setText(f"Intensità: {intensity}")

    def save_audio_file(self):
        try:
            filename =
f"registrazione_audio_{self.device_info[self.selected_device_index]['name']}.wav"
            sf.write(filename, self.audio_data, self.sample_rate,
subtype=f'PCM_{self.bit_depth}')
        except Exception as e:
            QMessageBox.critical(self, "Errore", f"Errore durante il salvataggio del file
audio: {str(e)}")

if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = AudioRecorderApp()
    window.show()
    sys.exit(app.exec_())
```

Process Monitor

Autore: Bocaletto Luca
Licenza: GPLv3

Process Monitor è una semplice applicazione Python per monitorare e gestire i processi in esecuzione sul tuo computer. Fornisce un'interfaccia grafica per visualizzare e interagire con i processi, semplificando il monitoraggio dell'utilizzo delle risorse di sistema.

Funzionalità

- Visualizza un elenco di processi in esecuzione con dettagli come PID, nome, utilizzo della CPU, utilizzo della memoria, stato, utente, orario di creazione e altro.
- Ordina l'elenco dei processi per diverse colonne.
- Termina i processi.
- Avvia nuovi processi.
- Personalizza le colonne visualizzate nell'elenco dei processi.
- Cerca processi specifici.

Source Code - Process Monitor

```
# Nome Software: Process Monitor
# Author: Bocaletto Luca
# License: GPLv3
# Importa le librerie necessarie
import tkinter as tk  # Per la GUI
from tkinter import ttk
import psutil  # Per ottenere informazioni sui processi
import subprocess  # Per avviare nuovi processi

# Dichiarazione delle variabili globali
colonna_ordinamento = 'name'  # La colonna iniziale per l'ordinamento
ordine_crescente = True  # Ordine iniziale ascendente
colonne_da_mostrare = ["PID", "Nome", "CPU", "Memoria", "Stato", "Utente",
"Avvio", "Priorità", "PID Genitore", "Cartella di Lavoro", "Uso Memoria"]
colonne_selezionate = {}  # Dizionario per tenere traccia delle colonne selezionate

# Funzione per mostrare i processi
def mostra_processi(filtro=""):
    # Ottieni una lista di oggetti processo con attributi specifici
    processi = list(psutil.process_iter(attrs=['pid', 'name', 'cpu_percent', 'memory_info',
'status', 'username', 'create_time', 'nice', 'ppid', 'cwd', 'memory_percent']))
```

```
    # Rimuovi i processi dalla visualizzazione
    for row in elenco_processi.get_children():
        elenco_processi.delete(row)

    # Funzione per ottenere il valore di una colonna per un processo
    def get_valore(processo, colonna):
        valore = processo.info.get(colonna)
        return valore if valore is not None else ""

    # Funzione per ottenere il valore 'nice' come intero
    def get_nice_as_int(processo):
        nice = processo.info.get('nice')
        return int(nice) if nice is not None else 0

    # Ordina i processi in base alla colonna di ordinamento e all'ordine
    processi_ordinati = sorted(processi, key=lambda p: get_nice_as_int(p) if
colonna_ordinamento == 'nice' else get_valore(p, colonna_ordinamento))
    if not ordine_crescente:
        processi_ordinati.reverse()

    # Aggiungi i processi alla lista
    for processo in processi_ordinati:
        pid = processo.info['pid']
        nome = processo.info['name']
        cpu_percent = processo.info['cpu_percent']
        memoria = processo.info['memory_info'].rss / (1024 * 1024)  # Converti in MB
        status = processo.info['status']
        username = get_valore(processo, 'username')
        create_time = processo.info['create_time']
        nice = processo.info['nice']
        ppid = processo.info['ppid']
        cwd = get_valore(processo, 'cwd')
        memory_percent = processo.info['memory_percent']

        # Filtra i processi in base al filtro inserito e aggiungili alla visualizzazione
        if filtro.lower() in nome.lower():
            elenco_processi.insert('', 'end', values=(pid, nome, f"{cpu_percent:.2f}%",
f"{memoria:.2f} MB", status, username, create_time, nice, ppid, cwd,
f"{memory_percent:.2f}%"))

# Funzione per cambiare l'ordinamento della colonna
def cambia_ordinamento(colonna):
    global ordine_crescente, colonna_ordinamento
    if colonna == colonna_ordinamento:
```

```
        ordine_crescente = not ordine_crescente
    else:
        colonna_ordinamento = colonna
        ordine_crescente = True
    mostra_processi(campo_ricerca.get())

# Funzione per terminare un processo
def termina_processo():
    selezionato = elenco_processi.selection()
    if selezionato:
        pid = int(elenco_processi.item(selezionato, 'values')[0])
        try:
            # Ottieni l'oggetto processo e termina il processo
            processo_da_terminare = psutil.Process(pid)
            processo_da_terminare.terminate()
            mostra_processi()  # Aggiorna la visualizzazione
        except psutil.NoSuchProcess:
            pass

# Funzione per avviare un nuovo processo
def avvia_processo():
    processo_da_avviare = campo_avvia.get()
    try:
        # Avvia il nuovo processo
        subprocess.Popen(processo_da_avviare, shell=True)
        campo_avvia.delete(0, tk.END)  # Cancella il campo di avvio
    except Exception as e:
        tk.messagebox.showerror("Errore", f"Impossibile avviare il processo:\
n{str(e)}")  # Mostra un messaggio di errore

# Funzione per mostrare le colonne selezionate
def mostra_colonne_selezionate():
    global colonne_da_mostrare
    colonne_da_mostrare = [col for col, var in colonne_selezionate.items() if var.get()
== 1]
    elenco_processi.config(columns=colonne_da_mostrare)  # Configura le colonne
visualizzate

    for col in colonne_da_mostrare:
        elenco_processi.heading(col, text=col)

    mostra_processi(campo_ricerca.get())  # Aggiorna la visualizzazione

# Funzione per aprire la finestra delle opzioni
def apri_finestra_opzioni():
```

```
    finestra_opzioni = tk.Toplevel(root)
    finestra_opzioni.title("Opzioni")

    colonne_frame = tk.Frame(finestra_opzioni)
    colonne_frame.grid(row=0, column=0, rowspan=4, pady=5, padx=10)
    tk.Label(colonne_frame, text="Colonne da mostrare:").grid(row=0, column=0,
columnspan=2)

    for i, col in enumerate(colonne_da_mostrare):
        var = tk.IntVar()
        if col in colonne_da_mostrare:
            var.set(1)
        colonne_selezionate[col] = var
        tk.Checkbutton(colonne_frame, text=col, variable=var).grid(row=i + 1,
column=0, columnspan=2, sticky="w")

    conferma_colonne_button = tk.Button(colonne_frame, text="Conferma",
command=mostra_colonne_selezionate)
    conferma_colonne_button.grid(row=i + 2, column=0, columnspan=2)

# Crea la finestra principale
root = tk.Tk()
root.title("Monitor Process")  # Imposta il titolo

titolo_label = tk.Label(root, text="Monitor Process")
titolo_label.grid(row=0, column=0, columnspan=5, sticky="w")  # Aggiungi un titolo

campo_ricerca = tk.Entry(root, width=20)
campo_ricerca.grid(row=1, column=0, columnspan=5, padx=5, pady=5,
sticky="we")  # Campo di ricerca

elenco_processi = ttk.Treeview(root, columns=colonne_da_mostrare,
show="headings")
elenco_processi.grid(row=2, column=0, columnspan=5, padx=5, pady=5,
sticky="news")  # Visualizzazione dei processi

scrollbar = ttk.Scrollbar(root, orient="vertical", command=elenco_processi.yview)
scrollbar.grid(row=2, column=5, sticky="ns")
elenco_processi.configure(yscrollcommand=scrollbar.set)  # Barra di scorrimento
verticale

scrollbar_x = ttk.Scrollbar(root, orient="horizontal",
command=elenco_processi.xview)
scrollbar_x.grid(row=3, column=0, columnspan=5, sticky="ew")
elenco_processi.configure(xscrollcommand=scrollbar_x.set)  # Barra di scorrimento
```

```
orizzontale

for col in colonne_da_mostrare:
    elenco_processi.heading(col, text=col, command=lambda c=col:
cambia_ordinamento(c))
    elenco_processi.column(col, width=50, minwidth=50, anchor="center")  #
Intestazioni e larghezza delle colonne

avvia_frame = tk.Frame(root)
avvia_frame.grid(row=4, column=0, columnspan=5, pady=10, sticky="w")  # Frame
per le azioni

campo_avvia = tk.Entry(avvia_frame, width=20)
campo_avvia.grid(row=0, column=0, padx=5, sticky="w")  # Campo per avviare
nuovi processi

avvia_button = tk.Button(avvia_frame, text="Avvia", command=avvia_processo)
avvia_button.grid(row=0, column=1, padx=5, sticky="w")  # Bottone per avviare un
nuovo processo

termina_button = tk.Button(avvia_frame, text="Termina Processo",
command=termina_processo)
termina_button.grid(row=0, column=2, padx=5, sticky="w")  # Bottone per terminare
un processo

aggiorna_button = tk.Button(avvia_frame, text="Aggiorna", command=lambda:
mostra_processi(campo_ricerca.get()))
aggiorna_button.grid(row=0, column=3, padx=5, sticky="w")  # Bottone per
aggiornare la visualizzazione dei processi

opzioni_button = tk.Button(avvia_frame, text="Opzioni",
command=apri_finestra_opzioni)
opzioni_button.grid(row=0, column=4, padx=5, sticky="w")  # Bottone per aprire la
finestra delle opzioni

root.grid_rowconfigure(2, weight=1)
root.columnconfigure(0, weight=1)

mostra_processi()  # Mostra i processi all'avvio

root.mainloop()  # Avvia l'applicazione
```

Password Manager

Password Manager è una semplice applicazione Python per la gestione sicura delle tue password. Utilizza PyQt5 per l'interfaccia grafica e SQLite per lo storage del database. Le password sono crittografate utilizzando la libreria di crittografia Fernet, garantendo la sicurezza dei tuoi dati sensibili.

Caratteristiche

- Archivia e gestisci le password per vari servizi.
- I dati vengono memorizzati in un database SQLite per un facile recupero.
- Le password vengono crittografate con Fernet per una maggiore sicurezza.
- Interfaccia utente grafica (GUI) facile da usare.

Source Code – Password Manager

```
# Software Name: Password Manager
# Author: Bocaletto Luca
# Site Web: https://www.elektronoide.it

import sys
import sqlite3
import os
from PyQt5.QtWidgets import QApplication, QMainWindow, QWidget,
QVBoxLayout, QLabel, QLineEdit, QPushButton, QMessageBox
from cryptography.fernet import Fernet, InvalidToken, InvalidSignature

# Definizione della classe principale PasswordManager
class PasswordManager(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Password Manager")
        self.setGeometry(100, 100, 400, 200)

        # Connessione al database SQLite per le password
        try:
            self.conn = sqlite3.connect("passwords.db")
            self.create_table()
        except sqlite3.Error as e:
            QMessageBox.critical(self, "Errore", "Impossibile connettersi al database: " +
str(e))
```

```
            sys.exit(1)

        # Carica o genera la chiave di crittografia
        self.key = self.load_or_generate_key()

        # Inizializza l'interfaccia utente
        self.init_ui()

    # Crea la tabella per le password nel database se non esiste già
    def create_table(self):
        try:
            cursor = self.conn.cursor()
            cursor.execute('''CREATE TABLE IF NOT EXISTS passwords
                              (id INTEGER PRIMARY KEY,
                               service TEXT NOT NULL,
                               username TEXT NOT NULL,
                               password TEXT NOT NULL)''')
            self.conn.commit()
        except sqlite3.Error as e:
            QMessageBox.critical(self, "Errore", "Impossibile creare la tabella: " + str(e))
            sys.exit(1)

    # Inizializza l'interfaccia utente Qt
    def init_ui(self):
        layout = QVBoxLayout()

        self.service_label = QLabel("Servizio:")
        self.service_input = QLineEdit()

        self.username_label = QLabel("Username:")
        self.username_input = QLineEdit()

        self.password_label = QLabel("Password:")
        self.password_input = QLineEdit()
        self.password_input.setEchoMode(QLineEdit.Password)

        self.save_button = QPushButton("Salva Password")
        self.save_button.clicked.connect(self.save_password)

        self.retrieve_button = QPushButton("Recupera Password")
        self.retrieve_button.clicked.connect(self.retrieve_password)

        layout.addWidget(self.service_label)
        layout.addWidget(self.service_input)
        layout.addWidget(self.username_label)
```

```
    layout.addWidget(self.username_input)
    layout.addWidget(self.password_label)
    layout.addWidget(self.password_input)
    layout.addWidget(self.save_button)
    layout.addWidget(self.retrieve_button)

    central_widget = QWidget()
    central_widget.setLayout(layout)
    self.setCentralWidget(central_widget)

# Carica o genera la chiave di crittografia e la salva in un file
def load_or_generate_key(self):
    key_file_path = "key.key"
    if os.path.exists(key_file_path):
        with open(key_file_path, "rb") as key_file:
            key = key_file.read()
    else:
        key = Fernet.generate_key()
        with open(key_file_path, "wb") as key_file:
            key_file.write(key)
    return key

# Crittografa una password usando la chiave
def encrypt_password(self, password, key):
    fernet = Fernet(key)
    return fernet.encrypt(password.encode())

# Decrittografa una password usando la chiave
def decrypt_password(self, encrypted_password, key):
    try:
        fernet = Fernet(key)
        return fernet.decrypt(encrypted_password).decode()
    except (InvalidToken, InvalidSignature) as e:
        print(f"Errore durante la decrittazione: {e}")
        return "Errore nella decrittazione"

# Salva una nuova password nel database
def save_password(self):
    service = self.service_input.text()
    username = self.username_input.text()
    password = self.password_input.text()

    if service and username and password:
        encrypted_password = self.encrypt_password(password, self.key)
```

```
            try:
                cursor = self.conn.cursor()
                cursor.execute("INSERT INTO passwords (service, username, password)
VALUES (?, ?, ?)", (service, username, encrypted_password))
                self.conn.commit()
                QMessageBox.information(self, "Successo", "Password salvata con
successo!")
                self.service_input.clear()
                self.username_input.clear()
                self.password_input.clear()
            except sqlite3.Error as e:
                QMessageBox.critical(self, "Errore", "Errore durante il salvataggio della
password: " + str(e))
        else:
            QMessageBox.warning(self, "Errore", "Tutti i campi sono obbligatori.")

    # Recupera e decrittografa una password dal database
    def retrieve_password(self):
        service = self.service_input.text()
        username = self.username_input.text()

        if service and username:
            try:
                cursor = self.conn.cursor()
                cursor.execute("SELECT password FROM passwords WHERE service=?
AND username=?", (service, username))
                row = cursor.fetchone()
                if row:
                    encrypted_password = row[0]
                    decrypted_password = self.decrypt_password(encrypted_password,
self.key)
                    self.password_input.setEchoMode(QLineEdit.Normal)  # Imposta il testo
della password visibile
                    self.password_input.setText(decrypted_password)
                else:
                    self.password_input.clear()
                    QMessageBox.warning(self, "Errore", "Password non trovata.")
            except sqlite3.Error as e:
                QMessageBox.critical(self, "Errore", "Errore durante il recupero della
password: " + str(e))
        else:
            QMessageBox.warning(self, "Errore", "Inserire il servizio e l'username.")

# Funzione principale per l'esecuzione del programma
if __name__ == "__main__":
```

```
app = QApplication(sys.argv)
window = PasswordManager()
window.show()
sys.exit(app.exec_())
```

Tris 2 Player

Questo è un gioco del Tris (Tic-Tac-Toe) a due giocatori realizzato in Python utilizzando la libreria Tkinter per l'interfaccia grafica.

Funzionalità

- Permette a due giocatori di alternarsi per effettuare mosse.
- Tiene traccia delle vittorie di entrambi i giocatori.
- Visualizza il punteggio dei giocatori.
- Mostra un messaggio di vittoria o pareggio quando viene raggiunta una condizione di fine partita.
- Consente di riavviare il gioco.

Come giocare

1. Esegui il gioco eseguendo il file `Tris2Player_Italian.py`.
2. Clicca su uno dei bottoni sulla griglia per effettuare una mossa.
3. Il gioco ti dirà quando hai vinto o pareggiato.
4. Clicca su "Riavvia" per iniziare una nuova partita.

```
# Nome del software: Tris 2 Player
# Author: Bocaletto Luca

# Importa la libreria Tkinter per la GUI e messagebox per i popup di messaggi.
import tkinter as tk
from tkinter import messagebox

# Classe principale per il gioco del Tris.
class TrisGame:
    # Variabili di classe per tenere traccia delle vittorie dei giocatori.
    player1_wins = 0
    player2_wins = 0

    def __init__(self):
        # Inizializza il gioco.
        self.current_player = "X"  # Inizia con il giocatore X.
        self.board = [" " for _ in range(9)]  # Inizializza una lista per la griglia vuota.
        self.buttons = []  # Lista di pulsanti sulla griglia.
```

```
    def setup_window(self):
        # Configura la finestra del gioco.
        self.window = tk.Tk()  # Crea una finestra Tkinter.
        self.window.title("Tris")  # Imposta il titolo della finestra.

        # Etichette per visualizzare il punteggio dei giocatori.
        self.label_player1 = tk.Label(self.window, text=f"Player 1 (X) Vittorie:
{TrisGame.player1_wins}")
        self.label_player1.grid(row=3, column=0, columnspan=3)

        self.label_player2 = tk.Label(self.window, text=f"Player 2 (O) Vittorie:
{TrisGame.player2_wins}")
        self.label_player2.grid(row=4, column=0, columnspan=3)

        # Crea la griglia di bottoni 3x3 per il gioco.
        for i in range(3):
            row = []
            for j in range(3):
                # Crea un pulsante con testo vuoto.
                button = tk.Button(self.window, text=" ", font=("normal", 20), width=10,
height=2,
                                   command=lambda row=i, col=j: self.on_button_click(row, col))
                button.grid(row=i, column=j)  # Posiziona il pulsante sulla griglia.
                row.append(button)  # Aggiunge il pulsante alla lista dei pulsanti.
            self.buttons.append(row)  # Aggiunge la riga di pulsanti alla lista dei pulsanti.

    def on_button_click(self, row, col):
        # Gestisce l'evento di clic su un pulsante.
        if self.board[row * 3 + col] == " ":
            self.board[row * 3 + col] = self.current_player  # Aggiorna la griglia con il
simbolo del giocatore corrente.
            text_color = "red" if self.current_player == "X" else "blue"  # Imposta il
colore del testo.
            self.buttons[row][col].config(text=self.current_player, fg=text_color)  #
Aggiorna il pulsante.

            if self.check_winner(row, col):
                # Verifica se il giocatore ha vinto.
                messagebox.showinfo("Vittoria!", f"Il giocatore {self.current_player} ha
vinto!")
                if self.current_player == "X":
                    TrisGame.player1_wins += 1  # Aggiorna il punteggio del giocatore X.
                else:
                    TrisGame.player2_wins += 1  # Aggiorna il punteggio del giocatore O.
```

```
            self.label_player1.config(text=f"Player 1 (X) Vittorie:
{TrisGame.player1_wins}")
            self.label_player2.config(text=f"Player 2 (O) Vittorie:
{TrisGame.player2_wins}")
            self.reset_board()  # Riavvia il gioco.
        else:
            if " " not in self.board:
                # Se non ci sono spazi vuoti sulla griglia, il gioco è un pareggio.
                messagebox.showinfo("Pareggio", "Il gioco è finito in pareggio.")
                self.reset_board()  # Riavvia il gioco.
            else:
                self.current_player = "O" if self.current_player == "X" else "X"  # Passa
al prossimo giocatore.

    def check_winner(self, row, col):
        # Controlla se il giocatore corrente ha vinto.
        directions = [(0, 1), (1, 0), (1, 1), (1, -1)]
        for dr, dc in directions:
            count = 1
            for i in range(1, 3):
                r = row + dr * i
                c = col + dc * i
                if 0 <= r < 3 and 0 <= c < 3 and self.board[r * 3 + c] == self.current_player:
                    count += 1
                else:
                    break
            for i in range(1, 3):
                r = row - dr * i
                c = col - dc * i
                if 0 <= r < 3 and 0 <= c < 3 and self.board[r * 3 + c] == self.current_player:
                    count += 1
                else:
                    break
            if count >= 3:
                return True  # Il giocatore ha vinto.

        return False  # Il giocatore non ha vinto.

    def reset_board(self):
        # Riavvia il gioco chiudendo la finestra attuale e creandone una nuova istanza.
        self.window.destroy()  # Chiude la finestra attuale.
        game = TrisGame()  # Crea una nuova istanza di TrisGame.
        game.run()  # Avvia il nuovo gioco.

    def run(self):
```

```
        # Avvia il gioco.
        self.setup_window()  # Configura la finestra.
        self.window.mainloop()  # Avvia il ciclo principale Tkinter.

if __name__ == "__main__":
    # Esegue il gioco quando il modulo viene eseguito come script principale.
    game = TrisGame()  # Crea un'istanza del gioco.
    game.run()  # Avvia il gioco.
```

Timed Shutdown PC

Timed Shutdown PC è un'applicazione Python basata su tkinter che ti consente di pianificare lo spegnimento, il riavvio o la sospensione del tuo computer in base all'orario specificato. È un'utility utile per automatizzare alcune azioni di spegnimento del computer.

Funzionalità principali

- **Visualizzazione dell'orario corrente:** L'app mostra un orologio digitale in tempo reale nella finestra principale.
- **Pianificazione delle azioni:** Puoi specificare un orario (nel formato HH:MM) e scegliere se spegnere, riavviare o sospendere il computer a quell'orario.
- **Validazione dell'orario:** L'app verifica se l'orario specificato è giusto e mostra un messaggio di errore se non lo è.
- **Esecuzione ritardata:** L'app attende fino all'orario specificato e quindi esegue l'azione selezionata.

Source Code - Timed Shutdown PC

```
# Software Name: Timed Shutdown PC
# Author: Bocaletto Luca
# Site Web: https://www.elektronoide.it
# License: GPLv3
# Importa il modulo tkinter per creare l'interfaccia utente grafica
import tkinter as tk
from tkinter import ttk
import os
import time
import threading
from tkinter import messagebox

# Definisci la classe principale dell'applicazione
class AppOrario:
```

```
    def __init__(self, root):
        self.root = root
        self.root.title("Spegnimento Programmato del PC")  # Imposta il titolo
dell'applicazione

        # Crea una label per visualizzare l'orario corrente
        self.etichetta_orario = tk.Label(root, text="", font=("Helvetica", 48))
        self.etichetta_orario.pack()

        # Avvia l'aggiornamento dell'orario
        self.avvia_orologio()

        # Crea l'interfaccia per la pianificazione delle azioni
        self.crea_interfaccia_pianificazione()

    def avvia_orologio(self):
        # Funzione per aggiornare l'orario in tempo reale
        def aggiorna_orologio():
            while True:
                orario_corrente = time.strftime("%H:%M:%S")
                self.etichetta_orario.config(text=orario_corrente)
                time.sleep(1)

        # Crea un thread per l'orologio che viene eseguito in background
        thread_orologio = threading.Thread(target=aggiorna_orologio)
        thread_orologio.daemon = True  # Il thread terminerà quando l'app verrà chiusa
        thread_orologio.start()

    def crea_interfaccia_pianificazione(self):
        # Crea un frame per l'interfaccia di pianificazione
        frame_pianificazione = ttk.LabelFrame(self.root, text="Pianificazione")
        frame_pianificazione.pack(padx=20, pady=20)

        # Etichetta per inserire l'orario
        etichetta_orario = ttk.Label(frame_pianificazione, text="Orario (HH:MM):")
        etichetta_orario.grid(row=0, column=0, padx=10, pady=10)

        # Casella di testo per inserire l'orario
        self.casella_orario = ttk.Entry(frame_pianificazione)
        self.casella_orario.grid(row=0, column=1, padx=10, pady=10)

        # Variabile per memorizzare l'azione selezionata
        self.variabile_azione = tk.StringVar()
        self.variabile_azione.set("Spegnimento")  # Imposta l'azione predefinita su
"Spegnimento"
```

```
        # Pulsanti per diverse azioni
        pulsante_spegnimento = ttk.Button(frame_pianificazione, text="Spegnimento",
command=self.azione_spegnimento)
        pulsante_spegnimento.grid(row=1, column=0, padx=10, pady=10)

        pulsante_riavvio = ttk.Button(frame_pianificazione, text="Riavvio",
command=self.azione_riavvio)
        pulsante_riavvio.grid(row=1, column=1, padx=10, pady=10)

        pulsante_sospensione = ttk.Button(frame_pianificazione, text="Sospensione",
command=self.azione_sospensione)
        pulsante_sospensione.grid(row=2, column=0, padx=10, pady=10)

        pulsante_pianifica = ttk.Button(frame_pianificazione, text="Pianifica",
command=self.pianifica_azione)
        pulsante_pianifica.grid(row=2, column=1, padx=10, pady=10)

    def pianifica_azione(self):
        # Funzione per pianificare un'azione
        orario_pianificato = self.casella_orario.get()

        if not orario_pianificato:
            # Se l'orario non è stato inserito, mostra un messaggio di errore
            messagebox.showerror("Errore", "Inserisci un orario valido prima di
pianificare un'azione.")
            return

        try:
            hh, mm = map(int, orario_pianificato.split(':'))
            orario_attuale = time.localtime()
            orario_pianificato = time.mktime((orario_attuale.tm_year,
orario_attuale.tm_mon, orario_attuale.tm_mday, hh, mm, 0, -1, -1, -1))
            orario_corrente = time.mktime(time.localtime())
            ritardo = orario_pianificato - orario_corrente

            if ritardo <= 0:
                # Se l'orario è nel passato, mostra un messaggio di errore
                messagebox.showerror("Errore", "Inserisci un orario futuro valido.")
            else:
                # Attendi il ritardo e quindi esegui l'azione pianificata
                self.root.after(int(ritardo * 1000), self.esegui_azione)
        except ValueError:
            # Se l'orario ha un formato non valido, mostra un messaggio di errore
            messagebox.showerror("Errore", "Formato orario non valido (HH:MM).")
```

```
    def esegui_azione(self):
        # Funzione per eseguire l'azione pianificata
        azione_selezionata = self.variabile_azione.get()
        if azione_selezionata == "Spegnimento":
            os.system("shutdown /s /f /t 0")  # Spegni il sistema
        elif azione_selezionata == "Riavvio":
            os.system("shutdown /r /f /t 0")  # Riavvia il sistema
        elif azione_selezionata == "Sospensione":
            os.system("shutdown /h /f")  # Sospendi il sistema

    def azione_spegnimento(self):
        self.variabile_azione.set("Spegnimento")

    def azione_riavvio(self):
        self.variabile_azione.set("Riavvio")

    def azione_sospensione(self):
        self.variabile_azione.set("Sospensione")

if __name__ == "__main__":
    # Crea la finestra principale dell'applicazione
    root = tk.Tk()
    app = AppOrario(root)
    root.mainloop()  # Avvia l'interfaccia utente principale
```

Browser Temporary Files Delete

Autore: Luca Bocaletto
Sito Web: https://www.elektronoide.it
Licenza: GPLv3 (Licenza Pubblica Generale GNU, versione 3)

Panoramica

Browser Temporary Files Delete è un'applicazione Python che aiuta gli utenti a eliminare i file temporanei dai browser web. Fornisce un'interfaccia utente amichevole per la selezione dei browser da scannerizzare e da cui eliminare i file temporanei. Il software rileva dinamicamente i percorsi di cache dei browser web comuni e consente all'utente di eseguire operazioni di pulizia con facilità.

Funzionalità

- Interfaccia utente realizzata con PyQt5, che offre un ambiente user-friendly.
- Capacità di scannerizzare ed eliminare i file temporanei dai browser web selezionati.
- Rilevamento dei percorsi di cache per browser web comuni, come Google Chrome, Mozilla

Firefox, Microsoft Edge, Opera e Safari.
- Opzione per visualizzare una finestra di dialogo "About" per saperne di più sull'applicazione.
- Gestione degli errori per informare l'utente di eventuali problemi riscontrati durante la pulizia dei file.

Utilizzo

1. Avviare l'applicazione.
2. Selezionare i browser web da pulire spuntando le caselle corrispondenti.
3. Fare clic sul pulsante "Avvia Scansione" per scannerizzare e visualizzare il numero totale di file temporanei trovati.
4. Fare clic sul pulsante "Elimina File" per rimuovere i file selezionati.
5. Utilizzare l'opzione "About" nel menu per visualizzare informazioni sull'applicazione.

Source Code – Browser Temprary Files Delete

```
# Software Name: Browser Temporary Files Delete
# Author: Bocaletto Luca
# Site Web: https://www.elektronoide.it
# License: GPLv3

import os
import sys
from PyQt5.QtWidgets import QApplication, QMainWindow, QWidget,
QVBoxLayout, QGroupBox, QCheckBox, QPushButton, QLabel, QMessageBox,
QMenuBar, QAction, QDialog
from PyQt5 import QtCore
import getpass

# Definizione della classe per la finestra About
class AboutDialog(QDialog):
    def __init__(self):
        super().__init__()

        # Imposta il titolo della finestra di dialogo
        self.setWindowTitle("About Browser Temporary Files Delete")
        self.setGeometry(200, 200, 300, 100)

        # Crea un layout verticale per la finestra di dialogo
        layout = QVBoxLayout()

        # Aggiungi un QLabel con le informazioni sull'applicazione
        about_label = QLabel("Browser Temporary Files Delete - \n\nAuthor: Bocaletto
```

```
Luca", self)
        layout.addWidget(about_label)

        # Imposta il layout per la finestra di dialogo
        self.setLayout(layout)

# Definizione della classe principale dell'applicazione
class BrowserCacheCleanerApp(QMainWindow):
    def __init__(self):
        super().__init__()

        # Ottieni il nome utente dell'utente attualmente loggato
        self.yourusername = getpass.getuser()

        # Elenco dei percorsi delle cache dei browser
        self.browsers = [
            {"name": "Google Chrome", "path":
f"C:/Users/{self.yourusername}/AppData/Local/Google/Chrome/User
Data/Default/Cache"},
            {"name": "Mozilla Firefox", "path":
f"C:/Users/{self.yourusername}/AppData/Local/Mozilla/Firefox/Profiles/
xxxxxxxx.default/cache2"},
            {"name": "Microsoft Edge", "path":
f"C:/Users/{self.yourusername}/AppData/Local/Microsoft/Edge/User
Data/Default/Cache"},
            {"name": "Opera", "path":
f"C:/Users/{self.yourusername}/AppData/Roaming/Opera Software/Opera
Stable/Cache"},
            {"name": "Safari", "path":
f"C:/Users/{self.yourusername}/AppData/Local/Apple Computer/Safari/Cache"}
        ]

        # Inizializza l'interfaccia utente
        self.initUI()

    def initUI(self):
        # Imposta il titolo della finestra
        self.setWindowTitle("Browser Temporary Files Delete")
        self.setGeometry(100, 100, 400, 300)

        # Crea una barra del menu in alto
        menubar = self.menuBar()
        about_action = QAction('About', self)
        about_action.triggered.connect(self.show_about_dialog)
        menubar.addAction(about_action)
```

```
# Crea un widget centrale per la finestra
central_widget = QWidget(self)
self.setCentralWidget(central_widget)

# Crea un layout verticale per organizzare gli elementi dell'interfaccia utente
layout = QVBoxLayout()

# Aggiungi un QLabel per il titolo
title_label = QLabel("Browser Temporary Files Delete", central_widget)
title_label.setAlignment(QtCore.Qt.AlignCenter)  # Allinea il testo al centro
layout.addWidget(title_label)

# Crea un gruppo di caselle di controllo per selezionare i browser
group_box = QGroupBox("Seleziona i Browser da pulire", central_widget)
group_layout = QVBoxLayout()

self.checkboxes = []

# Crea caselle di controllo per selezionare i browser
for browser in self.browsers:
    checkbox = QCheckBox(browser["name"], self)
    self.checkboxes.append(checkbox)
    group_layout.addWidget(checkbox)

group_box.setLayout(group_layout)

layout.addWidget(group_box)

# Etichetta per visualizzare il risultato
self.result_label = QLabel("", central_widget)
layout.addWidget(self.result_label)

# Pulsante per avviare la scansione
scan_button = QPushButton("Avvia Scan", central_widget)
scan_button.clicked.connect(self.scan_and_display)
layout.addWidget(scan_button)

# Pulsante per eliminare i file
delete_button = QPushButton("Elimina File", central_widget)
delete_button.clicked.connect(self.delete_files)
layout.addWidget(delete_button)

# Imposta il layout per il widget centrale
central_widget.setLayout(layout)
```

```
    # Funzione per mostrare la finestra di dialogo "About"
    def show_about_dialog(self):
        about_dialog = AboutDialog()
        about_dialog.exec_()

    # Funzione per eseguire la scansione e visualizzare il numero di file trovati
    def scan_and_display(self):
        num_files_found = 0
        for i, checkbox in enumerate(self.checkboxes):
            if checkbox.isChecked():
                browser = self.browsers[i]
                cache_path = browser["path"]
                num_files = self.count_files_in_directory(cache_path)
                num_files_found += num_files
        self.result_label.setText(f"Numero totale di file trovati: {num_files_found}")

    # Funzione per eliminare i file
    def delete_files(self):
        for i, checkbox in enumerate(self.checkboxes):
            if checkbox.isChecked():
                browser = self.browsers[i]
                cache_path = browser["path"]
                self.clean_browser_cache(cache_path, browser["name"])
        QMessageBox.information(self, "Eliminazione completata", "Eliminazione file
completata.")

    # Funzione per contare i file in una directory
    def count_files_in_directory(self, path):
        num_files = 0
        for root, dirs, files in os.walk(path):
            num_files += len(files)
        return num_files

    # Funzione per pulire la cache di un browser
    def clean_browser_cache(self, path, browser_name):
        try:
            for root, dirs, files in os.walk(path):
                for file in files:
                    file_path = os.path.join(root, file)
                    try:
                        os.remove(file_path)
                    except Exception as e:
                        print(f"Impossibile eliminare {file_path}: {e}")
            print(f"Cache di {browser_name} è stato pulito.")
```

```
        except Exception as e:
            print(f"Si è verificato un errore durante la pulizia {browser_name} cache:
{e}")

# Punto di ingresso dell'applicazione
if __name__ == '__main__':
    app = QApplication(sys.argv)
    window = BrowserCacheCleanerApp()
    window.show()
    sys.exit(app.exec_())
```

Send Remote Commands

Autore: Bocaletto Luca
Licenza: GPLv3

Descrizione

Send Remote Commands è un'applicazione Python sviluppata utilizzando la libreria GUI tkinter e il modulo socket. Permette agli utenti di connettersi a un server remoto, autenticarsi con un nome utente e una password e inviare comandi al server.

Caratteristiche

- Fornisce un'interfaccia utente grafica per interagire con un server remoto.
- Consente agli utenti di inserire l'indirizzo IP del server, la porta, il nome utente e la password per l'autenticazione.
- Gestisce l'autenticazione con il server remoto e visualizza lo stato della connessione.
- Permette agli utenti di inviare comandi al server e ricevere risposte.
- Offre una gestione degli errori per diverse problematiche di connessione e comandi.

Utilizzo

1. Avvia l'applicazione.
2. Inserisci l'indirizzo IP e la porta del server remoto.
3. Inserisci il tuo nome utente e la password.
4. Fai clic sul pulsante "Connetti al Server" per stabilire la connessione.
5. Una volta connessi, puoi inserire un comando nel campo "Comando".
6. Fai clic sul pulsante "Invia Comando" per inviare il comando al server.
7. La risposta dal server verrà visualizzata nella sezione "Risposta del Server".

Source Code – Invia Comandi Remoti

```
# Software Name: Send Remote Commands
```

```
# Author: Bocaletto Luca
# Site Web: https://www.elektronoide.it
# License: GPLv3

import tkinter as tk
import tkinter.messagebox as messagebox
import socket

# Creazione della finestra principale
window = tk.Tk()
window.title("Invia Comandi Remoti")

# Etichetta per il titolo
title_label = tk.Label(window, text="Invia Comandi Remoti", font=("Helvetica", 16))
title_label.pack()

# Etichette e campi di input per l'indirizzo IP del server, la porta, il nome utente e la
password
ip_label = tk.Label(window, text="Indirizzo IP del Server:")
ip_label.pack()
ip_entry = tk.Entry(window)
ip_entry.pack()

port_label = tk.Label(window, text="Porta del Server:")
port_label.pack()
port_entry = tk.Entry(window)
port_entry.pack()

username_label = tk.Label(window, text="Nome Utente:")
username_label.pack()
username_entry = tk.Entry(window)
username_entry.pack()

password_label = tk.Label(window, text="Password:")
password_label.pack()
password_entry = tk.Entry(window, show="*")  # Usa 'show' per nascondere la
password
password_entry.pack()

# Pulsante per connettersi al server (nella finestra principale)
connect_button = tk.Button(window, text="Connetti al Server")
connect_button.pack()

# Etichetta per lo stato della connessione (nella finestra principale)
status_label = tk.Label(window, text="")
```

```
status_label.pack()

# Etichetta e campo di input per inviare comandi (nella finestra principale)
command_label = tk.Label(window, text="Comando:")
command_label.pack()
command_entry = tk.Entry(window)
command_entry.pack()

# Pulsante per inviare comandi (nella finestra principale)
send_button = tk.Button(window, text="Invia Comando")
send_button.pack()

# Etichetta per la risposta del server (nella finestra principale)
response_label = tk.Label(window, text="Risposta del Server:")
response_label.pack()
response_text = tk.Text(window, height=10, width=40)
response_text.pack()
response_text.config(state=tk.DISABLED)

# Inizializzazione del socket client
client_socket = None

# Funzione per connettersi al server remoto con autenticazione
def connect_to_server():
    global client_socket
    server_ip = ip_entry.get()
    server_port_str = port_entry.get()
    username = username_entry.get()
    password = password_entry.get()

    try:
        server_port = int(server_port_str)
    except ValueError:
        messagebox.showerror("Errore", "La porta deve essere un numero intero.")
        return

    if not server_ip or not server_port_str or not username or not password:
        messagebox.showerror("Errore", "Inserisci un indirizzo IP valido, una porta, un
nome utente e una password.")
        return

    try:
        client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        client_socket.connect((server_ip, server_port))
        # Invia nome utente e password al server per l'autenticazione
```

```
        dati_autenticazione = f"{username}:{password}"
        client_socket.send(dati_autenticazione.encode())
        risposta = client_socket.recv(1024).decode()

        if risposta == "Autenticato":
            status_label.config(text="Connesso al server")
            connect_button.config(state=tk.DISABLED)
        else:
            status_label.config(text="Autenticazione fallita")
            client_socket.close()
    except Exception as e:
        status_label.config(text=f"Errore di connessione: {str(e)}")

# Pulsante per connettersi al server (nella finestra principale)
connect_button.config(command=connect_to_server)

# Funzione per inviare comandi al server remoto
def send_command():
    comando = command_entry.get()
    if comando:
        try:
            client_socket.send(comando.encode())
            risposta = client_socket.recv(1024).decode()
            response_text.config(state=tk.NORMAL)
            response_text.delete(1.0, tk.END)  # Cancella il testo esistente
            response_text.insert(tk.END, risposta)
            response_text.config(state=tk.DISABLED)
        except Exception as e:
            response_text.config(state=tk.NORMAL)
            response_text.delete(1.0, tk.END)
            response_text.insert(tk.END, f"Errore: {str(e)}")
            response_text.config(state=tk.DISABLED)

# Pulsante per inviare comandi (nella finestra principale)
send_button.config(command=send_command)

# Ciclo GUI principale
window.mainloop()
```

World Clock With Alarm

World Clock With Alarm è un'applicazione basata su Python che ti consente di monitorare l'orario in diversi fusi orari di tutto il mondo e impostare sveglie personalizzate. Questo software utilizza il modulo tkinter per l'interfaccia utente e

offre funzionalità come la visualizzazione dell'orario attuale in un fuso orario selezionato, nonché la possibilità di impostare sveglie in base a fusi orari specifici. Puoi anche selezionare un paese o una città e ottenere l'orario corrente in quel luogo. World Clock With Alarm è progettato per essere facile da usare e personalizzare secondo le tue esigenze.

Source Code – Orologio Mondiale con Allarme

```
# Name Software: World Clock With Alarm
# Author: Bocaletto Luca
# Site Web: https://www.elektronoide.it
# License: GPLv3
# Import necessary modules
# Importa i moduli necessari
import tkinter as tk
from tkinter import ttk, messagebox
import pytz
from datetime import datetime
import threading
import winsound

# Function to display the selected time for the chosen time zone
# Funzione per visualizzare l'orario selezionato per il fuso orario scelto
def show_selected_time():
    selection = countries_combobox.get()
    if selection:
        try:
            # Get the selected time zone and current time in that zone
            # Ottieni il fuso orario selezionato e l'orario corrente in quel fuso orario
            time_zone = pytz.timezone(selection)
            current_time = datetime.now(time_zone)

            # Update the time label with the current time in the selected time zone
            # Aggiorna l'etichetta dell'orario con l'orario corrente nel fuso orario
selezionato
            time_label.config(text=f'Time in {selection}: {current_time.strftime("%H:
%M:%S")}')
        except pytz.exceptions.UnknownTimeZoneError:
            # Display an error message if the time zone is unknown
            # Visualizza un messaggio di errore se il fuso orario è sconosciuto
            time_label.config(text=f'Fuso orario sconosciuto per {selection}')

    # Schedule the function to run again after 1 second
    # Programma l'esecuzione della funzione dopo 1 secondo
```

```
    app.after(1000, show_selected_time)
# Function to set an alarm
# Funzione per impostare una sveglia
def set_alarm():
    selected_time = alarm_time_entry.get()
    if not selected_time:
        return

    # Get the current time
    # Ottieni l'orario corrente
    current_time = datetime.now()

    # Extract the hour and minute from the selected time for the alarm
    # Estrai l'ora e il minuto dall'orario selezionato per la sveglia
    alarm_hour, alarm_minute = map(int, selected_time.split(":"))

    # Create a datetime object for the alarm time
    # Crea un oggetto datetime per l'orario della sveglia
    alarm_time = current_time.replace(hour=alarm_hour, minute=alarm_minute,
second=0, microsecond=0)

    selection = countries_combobox.get()
    if selection:
        try:
            # Get the selected time zone
            # Ottieni il fuso orario selezionato
            time_zone = pytz.timezone(selection)

            # Localize the alarm time to the selected time zone
            # Localizza l'orario della sveglia nel fuso orario selezionato
            alarm_time = time_zone.localize(alarm_time)

            # Calculate the time difference between the alarm time and current time
            # Calcola la differenza di tempo tra l'orario della sveglia e l'orario corrente
            difference = alarm_time - datetime.now(time_zone)

            if difference.total_seconds() > 0:
                # Function to play the alarm sound and show a message
                # Funzione per riprodurre il suono della sveglia e mostrare un messaggio
                def play_alarm():
                    winsound.Beep(500, 1000)
                    messagebox.showinfo("Sveglia", f"Sveglia impostata per
{selected_time} in {selection}")

                # Schedule the alarm function to run after the time difference
```

```
            # Programma l'esecuzione della funzione della sveglia dopo la differenza di tempo
            threading.Timer(difference.total_seconds(), play_alarm).start()

            # Display a success message with green text
            # Visualizza un messaggio di successo con testo verde
            alarm_status_label.config(text=f'Sveglia impostata per {selected_time} in {selection}', fg="green")
        else:
            # Display a message in red text indicating that the selected time has already passed
            # Visualizza un messaggio con testo rosso che indica che l'orario selezionato è già trascorso
            alarm_status_label.config(text=f'L\'orario selezionato è già trascorso.', fg="red")
    except pytz.exceptions.UnknownTimeZoneError:
        # Display an error message in red text for an unknown time zone
        # Visualizza un messaggio di errore con testo rosso per un fuso orario sconosciuto
        alarm_status_label.config(text=f'Fuso orario sconosciuto per {selection}', fg="red")
else:
    # Display a message in red text indicating that no time zone was selected
    # Visualizza un messaggio con testo rosso che indica che nessun fuso orario è stato selezionato
    alarm_status_label.config(text='Si prega di selezionare un fuso orario', fg="red")
# Create the main application window (Crea la finestra principale dell'applicazione)
app = tk.Tk()
app.title('Orologio Mondiale con Sveglia')

# Create and configure UI elements (Crea e configura gli elementi dell'interfaccia utente)
title_label = tk.Label(app, text='Orologio Mondiale con Sveglia', font=("Helvetica", 16))
title_label.pack(pady=10)

lbl_countries = tk.Label(app, text='Seleziona un paese:')
lbl_countries.pack()

# Populate the time zone selection combobox with available time zones
# Popola la casella combinata di selezione del fuso orario con i fusi orari disponibili
countries = pytz.all_timezones
countries_combobox = ttk.Combobox(app, values=countries)
countries_combobox.pack()
```

```
show_selected_button = tk.Button(app, text='Mostra Ora',
command=show_selected_time)
show_selected_button.pack()

time_label = tk.Label(app, text='')
time_label.pack()

set_alarm_button = tk.Button(app, text='Imposta Sveglia', command=set_alarm)
set_alarm_button.pack()

alarm_time_label = tk.Label(app, text='Inserisci l\'orario della sveglia (HH:MM):')
alarm_time_label.pack()

alarm_time_entry = tk.Entry(app)
alarm_time_entry.pack()

alarm_status_label = tk.Label(app, text='')
alarm_status_label.pack()

# Periodically update the displayed time (Aggiorna periodicamente l'orario
visualizzato)
app.after(1000, show_selected_time)

# Start the main event loop (Avvia il ciclo degli eventi principale)
app.mainloop()
```

Atomic Time Synch

Autore: Bocaletto Luca
Sito Web: https://www.elektronoide.it
Licenza: GPLv3

Panoramica

Atomic Time Synch è un'applicazione Python che consente di sincronizzare l'orario del sistema con una fonte di tempo atomico utilizzando il Protocollo di Tempo di Rete (NTP). Fornisce opzioni di sincronizzazione temporale sia manuali che automatiche e consente di scegliere tra diversi server NTP.

Funzionalità

- Aggiornamento manuale dell'orario di sistema.
- Sincronizzazione automatica dell'orario di sistema a intervalli specificati.
- Scelta da un elenco di server NTP per la sincronizzazione dell'orario.
- Visualizzazione dell'orario atomico e dell'orario locale.

Source Code - Atomic Time Synch

```
# Software Name: Atomic Time Synch
# Author: Bocaletto Luca

# Site Web: https://www.elektronoide.it

# License: GPLv3



import sys

import ntplib

import subprocess

from datetime import datetime

from PyQt5.QtCore import QTimer

from PyQt5.QtWidgets import QApplication, QMainWindow, QVBoxLayout, QWidget, QLabel, QPushButton, QComboBox



# Definizione della classe principale dell'applicazione

class AtomicTimeApp(QMainWindow):

    def __init__(self):

        super().__init__()

        self.setWindowTitle("Atomic Time Synch")  # Imposta il titolo della finestra

        self.setGeometry(100, 100, 400, 250)  # Imposta la dimensione della finestra
```

```
        self.central_widget = QWidget()
        self.setCentralWidget(self.central_widget)

        self.layout = QVBoxLayout()  # Crea un layout principale per la finestra

        # Creazione e configurazione del Label per il titolo
        self.title_label = QLabel("Atomic Time Synch")  # Crea un Label per il titolo
        self.title_label.setStyleSheet("font-size: 20px; text-align: center;")  # Imposta la dimensione dei caratteri e l'allineamento
        self.layout.addWidget(self.title_label)

        # Creazione del pulsante per l'aggiornamento manuale
        self.update_button = QPushButton("Aggiorna Ora")
        self.update_button.setStyleSheet("font-size: 14px;")  # Imposta la dimensione dei caratteri
        self.update_button.clicked.connect(self.update_time)  # Collega il pulsante all'aggiornamento
        self.layout.addWidget(self.update_button)

        # Creazione e configurazione dei controlli per l'aggiornamento automatico
        self.ntp_label = QLabel("Seleziona un Server NTP:")
        self.ntp_label.setStyleSheet("font-size: 14px;")  # Imposta la dimensione dei caratteri
```

```
        self.layout.addWidget(self.ntp_label)


        self.ntp_combo = QComboBox()
        self.ntp_combo.addItem("pool.ntp.org", "pool.ntp.org")
        self.ntp_combo.addItem("time.nist.gov", "time.nist.gov")
        # Aggiungi altri server NTP qui
        self.layout.addWidget(self.ntp_combo)


        self.interval_label = QLabel("Seleziona Intervallo di Aggiornamento Automatico (secondi):")
        self.interval_label.setStyleSheet("font-size: 14px;")  # Imposta la dimensione dei caratteri
        self.layout.addWidget(self.interval_label)


        self.interval_combo = QComboBox()
        for seconds in range(1, 11):
            self.interval_combo.addItem(f"{seconds} secondi", seconds)
        self.layout.addWidget(self.interval_combo)


        self.start_auto_update_button = QPushButton("Avvia Aggiornamento Automatico")
        self.start_auto_update_button.setStyleSheet("font-size: 14px;")  # Imposta la dimensione dei caratteri
        self.start_auto_update_button.clicked.connect(self.start_auto_update)
```

```
        self.layout.addWidget(self.start_auto_update_button)


        self.stop_auto_update_button = QPushButton("Ferma Aggiornamento
Automatico")
        self.stop_auto_update_button.setStyleSheet("font-size: 14px;")  # Imposta la
dimensione dei caratteri
        self.stop_auto_update_button.clicked.connect(self.stop_auto_update)
        self.layout.addWidget(self.stop_auto_update_button)
        self.stop_auto_update_button.setEnabled(False)


        # Creazione del pulsante per l'aggiornamento dell'orario di sistema
        self.sync_system_time_button = QPushButton("Aggiorna Ora di Sistema")
        self.sync_system_time_button.setStyleSheet("font-size: 14px;")  # Imposta la
dimensione dei caratteri
        self.sync_system_time_button.clicked.connect(self.sync_system_time)
        self.layout.addWidget(self.sync_system_time_button)


        # Creazione del Label per visualizzare l'orario
        self.time_label = QLabel()
        self.time_label.setStyleSheet("font-size: 16px;")  # Imposta la dimensione dei
caratteri per l'orario
        self.layout.addWidget(self.time_label)


        # Impostazione del layout principale
```

```
        self.central_widget.setLayout(self.layout)


        self.timer = None  # Variabile per gestire il timer dell'aggiornamento automatico
        self.auto_update_enabled = False  # Flag per indicare se l'aggiornamento automatico è attivo


        self.update_time()  # Aggiorna l'orario iniziale


    # Funzione per aggiornare l'orario manualmente
    def update_time(self):
        try:
            atomic_time = get_atomic_time()  # Ottieni l'orario atomico
            self.time_label.setText(f"Ora Atomica: {atomic_time} -- Ora Locale: {get_local_time()}")
        except Exception as e:
            self.show_error_message(f"Errore: {e}")


    # Funzione per mostrare un messaggio di errore
    def show_error_message(self, error_message):
        # Implementa la finestra di errore qui (da fare)
        pass


    # Funzione per sincronizzare l'orario di sistema con un server NTP
```

```
    def sync_system_time(self):
        selected_ntp_server = self.ntp_combo.currentData()
        try:
            sync_system_time_with_ntp(selected_ntp_server)
            self.show_success_message("Orario del sistema aggiornato con successo.")
        except Exception as e:
            self.show_error_message(f"Errore nell'aggiornamento dell'orario del sistema: {e}")

    # Funzione per mostrare un messaggio di successo
    def show_success_message(self, message):
        # Implementa la finestra di successo qui (da fare)
        pass

    # Funzione per avviare l'aggiornamento automatico
    def start_auto_update(self):
        interval_seconds = self.interval_combo.currentData()
        if not self.auto_update_enabled:
            self.timer = QTimer(self)
            self.timer.timeout.connect(self.update_time)
            self.timer.start(interval_seconds * 1000)
            self.auto_update_enabled = True
            self.start_auto_update_button.setEnabled(False)
```

```
            self.stop_auto_update_button.setEnabled(True)

    # Funzione per fermare l'aggiornamento automatico
    def stop_auto_update(self):
        if self.auto_update_enabled:
            self.timer.stop()
            self.auto_update_enabled = False
            self.start_auto_update_button.setEnabled(True)
            self.stop_auto_update_button.setEnabled(False)

# Funzione per ottenere l'orario atomico da un server NTP
def get_atomic_time():
    try:
        c = ntplib.NTPClient()
        response = c.request('pool.ntp.org')
        atomic_time = datetime.fromtimestamp(response.tx_time)
        return atomic_time.strftime('%Y-%m-%d %H:%M:%S')
    except Exception as e:
        raise Exception("Impossibile recuperare l'orario atomico")

# Funzione per ottenere l'orario locale
def get_local_time():
```

```
    return datetime.now().strftime('%Y-%m-%d %H:%M:%S')


# Funzione per sincronizzare l'orario di sistema con un server NTP
def sync_system_time_with_ntp(ntp_server):
    try:
        subprocess.run(["w32tm", "/resync", "/rediscover", "/nowait"])
    except Exception as e:
        raise Exception("Impossibile sincronizzare l'orario del sistema")


if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = AtomicTimeApp()
    window.show()
    sys.exit(app.exec_())
```

Paint Free

Autore: Luca Bocaletto
Sito Web: www.elektronoide.it
Licenza: GPLv3

Paint Free è un'applicazione Python che offre un'interfaccia grafica per il disegno digitale. Questo strumento intuitivo consente agli utenti di esplorare la loro creatività attraverso il digitale, offrendo una vasta gamma di funzionalità di disegno.

Caratteristiche principali:

- **Strumenti di disegno personalizzabili:** Scegli il colore e la dimensione del pennello che si adattano meglio al tuo stile artistico.

- **Importa immagini:** Integra facilmente immagini dalla tua raccolta per aggiungere dettagli o ispirazione ai tuoi disegni.
- **Progetti su misura:** Crea nuovi progetti con dimensioni personalizzate per adattarli alle tue esigenze artistiche.
- **Esporta il tuo lavoro:** Salva i tuoi disegni come file PNG per conservarli o condividerli con altri.

Source Code – Paint Free

```
# Nome del software: Paint Free

# Autore: Luca Bocaletto

# Sito Web: https://www.elektronoide.it

# Licenza: GPLv3


import tkinter as tk

from tkinter.colorchooser import askcolor

from tkinter import filedialog, simpledialog

from PIL import Image, ImageDraw, ImageTk


class DrawingApp:

    def __init__(self, root):

        self.root = root

        self.root.title("Paint Free")


        self.canvas = tk.Canvas(root, bg="white", width=800, height=600)

        self.canvas.pack()
```

```
        self.canvas.bind("<Button-1>", self.start_drawing)
        self.canvas.bind("<B1-Motion>", self.draw)
        self.canvas.bind("<ButtonRelease-1>", self.stop_drawing)

        self.pen_color = "black"
        self.pen_size = 2
        self.drawing = False
        self.last_x, self.last_y = None, None

        # Contenitore per i pulsanti in alto
        self.button_container = tk.Frame(root)
        self.button_container.pack(side=tk.TOP, fill=tk.X)

        self.button_color = tk.Button(self.button_container, text="Choose Color",
command=self.choose_color)
        self.button_color.pack(side=tk.LEFT)

        self.button_pen_size = tk.Scale(self.button_container, label="Pen Size",
from_=1, to=10, orient="horizontal")
        self.button_pen_size.pack(side=tk.LEFT)
        self.button_pen_size.set(self.pen_size)
        self.button_pen_size.bind("<Motion>", self.update_pen_size)
```

```
        self.button_clear = tk.Button(self.button_container, text="Clear",
command=self.clear_canvas)

        self.button_clear.pack(side=tk.LEFT)



        self.button_add_image = tk.Button(self.button_container, text="Add Image",
command=self.add_image)

        self.button_add_image.pack(side=tk.LEFT)



        self.button_new_project = tk.Button(self.button_container, text="New Project",
command=self.create_new_project)

        self.button_new_project.pack(side=tk.LEFT)



        self.button_save_drawing = tk.Button(self.button_container, text="Save
Drawing", command=self.save_drawing)

        self.button_save_drawing.pack(side=tk.LEFT)



        self.active_image = None

        self.draw_on_image = False



    def start_drawing(self, event):

        if not self.draw_on_image:

            x, y = event.x, event.y

            self.drawing = True

            self.last_x, self.last_y = x, y
```

```
def draw(self, event):
    if self.drawing:
        x, y = event.x, event.y
        if self.last_x and self.last_y:
            self.canvas.create_line(
                self.last_x,
                self.last_y,
                x,
                y,
                fill=self.pen_color,
                width=self.pen_size
            )
            self.last_x, self.last_y = x, y

def stop_drawing(self, event):
    self.drawing = False

def choose_color(self):
    color = askcolor()[1]
    if color:
        self.pen_color = color
```

```
    def update_pen_size(self, event):
        self.pen_size = self.button_pen_size.get()

    def clear_canvas(self):
        self.canvas.delete("all")

    def add_image(self):
        file_path = filedialog.askopenfilename(filetypes=[("Image files", "*.png;*.jpg;*.jpeg;*.gif")])
        if file_path:
            image = Image.open(file_path)
            self.active_image = ImageTk.PhotoImage(image)
            self.canvas.create_image(0, 0, anchor=tk.NW, image=self.active_image)
            self.draw_on_image = True

    def create_new_project(self):
        new_width = simpledialog.askinteger("New Project", "Enter width (in pixels):", parent=self.root, minvalue=1)
        new_height = simpledialog.askinteger("New Project", "Enter height (in pixels):", parent=self.root, minvalue=1)
        if new_width and new_height:
            self.canvas.config(width=new_width, height=new_height)
```

```
    def save_drawing(self):
        file_path = filedialog.asksaveasfilename(defaultextension=".png", filetypes=[("PNG files", "*.png")])
        if file_path:
            self.canvas.postscript(file=file_path, colormode="color")

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

Search Files

Search Files è un'applicazione desktop sviluppata in Python che consente agli utenti di cercare file all'interno di una directory specifica. Questa applicazione offre un'interfaccia utente intuitiva e funzionale per semplificare il processo di ricerca di file.

Funzionalità Principali

- Ricerca di file per nome all'interno di una directory specifica.
- Visualizzazione dei risultati della ricerca in una tabella con le colonne "File" e "Percorso".
- Possibilità di aprire direttamente i file trovati dall'applicazione.
- Interfaccia utente semplice e intuitiva per un'esperienza utente ottimale.
- Gestione degli errori e feedback all'utente per un utilizzo senza intoppi.

Come Usare l'Applicazione

1. Esegui il file `search_files.py` per avviare l'applicazione.

2. Nella finestra principale, inserisci il nome del file che desideri cercare nel campo "Nome del file".

3. Fai clic sul pulsante "Sfoglia" per selezionare la directory in cui effettuare la ricerca o inserisci manualmente il percorso nella casella "Directory".

4. Premi il pulsante "Cerca" per avviare la ricerca dei file.
5. I risultati della ricerca verranno visualizzati nella tabella sottostante con i nomi dei file e i percorsi corrispondenti.
6. Per aprire un file dalla lista dei risultati, selezionalo e clicca sul pulsante "Apri File".

Source Code – Search Files

```
# Nome del software: Search Files
# Autore: Luca Bocaletto
# Sito Web: https://www.elektronoide.it
# Licenza: GPLv3


# Importa le librerie necessarie
import os
import tkinter as tk
from tkinter import filedialog
from tkinter import ttk


# Funzione per cercare i file
def search_files():
    # Ottieni il nome del file e la directory dalla GUI
    file_name = entry_filename.get()
    directory = entry_directory.get()
```

```
# Verifica se il nome del file è vuoto
if not file_name:
    result_tree.delete(*result_tree.get_children())
    result_tree.insert("", "end", values=("Inserisci un nome di file valido.", ""))
    return

# Verifica se la directory esiste
if not os.path.exists(directory):
    result_tree.delete(*result_tree.get_children())
    result_tree.insert("", "end", values=("La directory selezionata non esiste.", ""))
    return

# Avvia la barra di avanzamento
progress_bar.start()

# Pulisce i risultati precedenti nella tabella
result_tree.delete(*result_tree.get_children())

# Crea una lista per memorizzare i file trovati
found_files = []

# Utilizza os.walk per esplorare la directory e i suoi sotto-directory
```

```
    for root, dirs, files in os.walk(directory):
        for file in files:
            # Verifica se il nome del file cercato è presente nel nome del file corrente
            if file_name in file:
                file_path = os.path.join(root, file)
                found_files.append((file, file_path))

    # Arresta la barra di avanzamento
    progress_bar.stop()

    # Mostra i risultati nella tabella
    if not found_files:
        result_tree.insert("", "end", values=("Nessun file trovato.", ""))
    else:
        for file, file_path in found_files:
            result_tree.insert("", "end", values=(file, file_path))

# Funzione per sfogliare e selezionare una directory
def browse_directory():
    directory = filedialog.askdirectory()
    entry_directory.delete(0, tk.END)
    entry_directory.insert(0, directory)
```

```
# Funzione per aprire il file selezionato
def open_file():
    selected_item = result_tree.selection()
    if selected_item:
        item = result_tree.item(selected_item, "values")
        file_path = item[1]
        if os.path.isfile(file_path):
            os.startfile(file_path)

# Creazione della finestra principale dell'app
app = tk.Tk()
app.title("Cerca File")

# Etichetta per il nome del file
label_filename = tk.Label(app, text="Nome del file:")
label_filename.pack()

# Campo di input per il nome del file
entry_filename = tk.Entry(app)
entry_filename.pack()
```

```
# Etichetta per la directory
label_directory = tk.Label(app, text="Directory:")
label_directory.pack()

# Campo di input per la directory
entry_directory = tk.Entry(app)
entry_directory.pack()

# Pulsante per la selezione della directory
browse_button = tk.Button(app, text="Sfoglia", command=browse_directory)
browse_button.pack()

# Pulsante per avviare la ricerca
search_button = tk.Button(app, text="Cerca", command=search_files)
search_button.pack()

# Barra di avanzamento
progress_bar = ttk.Progressbar(app, mode="indeterminate")
progress_bar.pack()

# Creazione di una tabella per i risultati con colonne "File" e "Percorso"
result_tree = ttk.Treeview(app, columns=("File", "Percorso"), show="headings")
```

result_tree.heading("File", text="File")

result_tree.heading("Percorso", text="Percorso")

Posiziona la tabella nella finestra

result_tree.pack(fill="both", expand=True)

Pulsante per aprire il file selezionato

open_button = tk.Button(app, text="Apri File", command=open_file)

open_button.pack()

Avvia l'applicazione

app.mainloop()

Wikipedia Search App

Wikipedia Search App è un'applicazione desktop sviluppata in Python che consente agli utenti di effettuare ricerche su Wikipedia utilizzando un'interfaccia grafica intuitiva.

Funzionalità Principali

- Ricerca di articoli su Wikipedia in base a una parola chiave.
- Selezione della lingua per la ricerca tra una vasta gamma di opzioni.
- Visualizzazione dei risultati direttamente nell'applicazione.

Utilizzo

1. Esegui il file `wikipedia_search.py` per avviare l'applicazione.
2. Inserisci una parola chiave nella casella di ricerca.
3. Seleziona la lingua desiderata dall'elenco a discesa.
4. Fai clic sul pulsante "Cerca" per ottenere i risultati.

Source Code – Wikipedia Search

```
# Nome del Software: Wikipedia Search

# Autore: Bocaletto Luca

# Sito Web: https://www.elektronoide.it

# Licenza: GPLv3


# Importa le librerie necessarie

import tkinter as tk

from tkinter import ttk

import wikipediaapi


# Imposta un user agent personalizzato per le richieste

user_agent = "MyWikipediaSearchApp/1.0 (YourEmailAddress@example.com)"


# Definisci una funzione per effettuare la ricerca nella Wikipedia

def search_wikipedia():

    query = entry.get()  # Ottieni la query dalla casella di testo

    selected_language = language_var.get()  # Ottieni la lingua selezionata

    language_mapping = {

        "Italiano": "it",

        "Inglese": "en",

        "Francese": "fr",
```

```
        "Spagnolo": "es",
        "Tedesco": "de",
        "Portoghese": "pt",
        "Olandese": "nl",
        "Arabo": "ar",
        "Cinese": "zh",
        "Giapponese": "ja",
        "Russo": "ru",
        "Rumeno": "ro",
        "Albanese": "sq"
    }
    selected_language_code = language_mapping.get(selected_language, "en")  # Ottieni il codice della lingua
    wiki_wiki = wikipediaapi.Wikipedia(
        language=selected_language_code,
        extract_format=wikipediaapi.ExtractFormat.WIKI,
        user_agent=user_agent
    )
    page = wiki_wiki.page(query)
    result_text.config(state='normal')
    result_text.delete(1.0, tk.END)
    result_text.insert(tk.END, page.text)
    result_text.config(state='disabled')
```

```
# Inizializza la finestra principale dell'applicazione
root = tk.Tk()
root.title("Wikipedia Search")

# Aumenta la dimensione del carattere per l'etichetta del titolo
title_label = ttk.Label(root, text="Wikipedia Search", font=("Helvetica", 16))
title_label.grid(row=0, column=0, columnspan=2, pady=(10, 20))

# Etichetta per la casella di testo di input
label = ttk.Label(root, text="Inserisci una parola e cerca su Wikipedia:",
font=("Helvetica", 12))
label.grid(row=1, column=0, columnspan=2)

# Casella di testo per l'inserimento della query
entry = ttk.Entry(root, font=("Helvetica", 12))
entry.grid(row=2, column=0, columnspan=2)

# Menù a discesa per la selezione della lingua
language_var = tk.StringVar()
language_var.set("Inglese")  # Imposta l'inglese come lingua predefinita
languages = [
    "Italiano", "Inglese", "Francese", "Spagnolo", "Tedesco", "Portoghese",
```

```
    "Olandese", "Arabo", "Cinese", "Giapponese", "Russo", "Rumeno", "Albanese"
]
language_menu = ttk.OptionMenu(root, language_var, *languages)
language_menu.grid(row=3, column=0, columnspan=2)

# Bottone di ricerca
search_button = ttk.Button(root, text="Cerca", command=search_wikipedia)
search_button.grid(row=4, column=0, columnspan=2)

# Utilizza il layout a griglia per il widget result_text
result_text = tk.Text(root, height=20, width=40, state='disabled', font=("Helvetica", 12))
result_text.grid(row=5, column=0, columnspan=2, padx=10, pady=10, sticky="nsew")

# Configura la griglia per consentire il ridimensionamento del widget Text con la finestra
root.grid_rowconfigure(5, weight=1)
root.grid_columnconfigure(0, weight=1)

# Avvia l'applicazione
root.mainloop()
```

Multiple Wave Generator

Autore: Bocaletto Luca **Licenza:** GPLv3

Multiple Wave Generator è un software versatile per la generazione di audio che ti consente di creare e manipolare segnali audio con facilità. Che tu sia un musicista, un sound designer o semplicemente curioso delle forme d'onda, questa applicazione offre un'interfaccia utente intuitiva per generare ed esplorare vari tipi di onde sonore.

Funzionalità Principali

- **Supporto Multi-Generatore:** Crea fino a tre generatori audio indipendenti, ciascuno con le proprie impostazioni.
- **Selezione della Forma d'Onda:** Scegli tra forme d'onda popolari come Sine, Square, Triangle, Sawtooth o genera rumore casuale.
- **Controllo della Frequenza:** Regola la frequenza del suono generato per creare toni alla tua tonalità desiderata.
- **Regolazione del Volume:** Ottimizza il livello del volume per ciascun generatore per ottenere il bilanciamento audio perfetto.
- **Riproduzione Audio in Tempo Reale:** Avvia e interrompi la riproduzione audio a tua discrezione.
- **Visualizzazione della Forma d'Onda:** Visualizza la forma d'onda selezionata in tempo reale.
- **Salva l'Audio:** Salva l'audio generato come file WAV per utilizzi futuri.

Source Code – Multiple Wave Generator

```
# Software Name: Triple Multiple Wave Generator

# Author: Bocaletto Luca

# License: GPLv3



import sys

import numpy as np

import pyaudio

import wave

import threading
```

```
from PyQt5.QtWidgets import QApplication, QMainWindow, QSlider,
QPushButton, QVBoxLayout, QWidget, QLabel, QComboBox

from PyQt5.QtCore import Qt

from PyQt5.QtGui import QPalette, QColor


# Classe per generare l'audio

class AudioGenerator:

    def __init__(self, generator_id, frequency=440.0, volume=0.5, waveform='Sine'):

        # Inizializza i parametri dell'oggetto

        self.generator_id = generator_id  # Identificatore del generatore

        self.frequency = frequency  # Frequenza dell'onda sonora (default: 440 Hz)

        self.volume = volume  # Volume dell'onda sonora (default: 0.5)

        self.waveform = waveform  # Tipo di forma d'onda (default: 'Sine')

        self.audio_stream = None  # Oggetto per la riproduzione dell'audio

        self.playing = False  # Indica se l'audio è in riproduzione o no


    # Avvia la riproduzione dell'audio

    def start(self):

        p = pyaudio.PyAudio()

        self.audio_stream = p.open(format=pyaudio.paFloat32, channels=1,
rate=int(self.frequency * 10), output=True,

                                   stream_callback=self.callback)

        self.playing = True
```

```
    # Ferma la riproduzione dell'audio
    def stop(self):
        if self.audio_stream is not None:
            self.audio_stream.stop_stream()
            self.audio_stream.close()
            self.playing = False

    # Callback per generare i dati dell'audio in base alla forma d'onda selezionata
    def callback(self, in_data, frame_count, time_info, status):
        if self.waveform == 'Sine':
            waveform = np.sin(2 * np.pi * np.arange(frame_count) * self.frequency / 44100.0)
        elif self.waveform == 'Square':
            waveform = np.sign(np.sin(2 * np.pi * np.arange(frame_count) * self.frequency / 44100.0))
        elif self.waveform == 'Triangle':
            waveform = np.abs(2 * (np.arange(frame_count) * self.frequency / 44100.0 - np.floor(0.5 + np.arange(frame_count) * self.frequency / 44100.0)))
        elif self.waveform == 'Sawtooth':
            waveform = 2 * (np.arange(frame_count) * self.frequency / 44100.0 - np.floor(0.5 + np.arange(frame_count) * self.frequency / 44100.0))
        else:
            waveform = np.random.uniform(-1, 1, frame_count)
```

```
        data = (self.volume * waveform).astype(np.float32).tobytes()
        return (data, pyaudio.paContinue)

# Classe principale dell'app
class AudioGeneratorApp(QMainWindow):
    def __init__(self):
        super().__init__()
        self.generators = []  # Lista dei generatori audio
        self.init_ui()
        self.setWindowTitle("Triple Multiple Wave Generator")  # Imposta il titolo della finestra

    # Inizializza l'interfaccia utente
    def init_ui(self):
        central_widget = QWidget()
        self.setCentralWidget(central_widget)

        layout = QVBoxLayout()
        central_widget.setLayout(layout)

        # Aggiungi il titolo nella parte superiore con il colore del testo bianco
        title_label = QLabel("Triple Multiple Wave Generator")
```

```
        title_label.setStyleSheet("font-size: 20px; color: white; padding: 10px; background-color: indigo;")  # Imposta il colore del testo e il padding

        layout.addWidget(title_label)


        for i in range(3):

            generator = AudioGenerator(i)  # Crea un nuovo generatore audio

            self.generators.append(generator)  # Aggiunge il generatore alla lista


            label = QLabel(f'Generatore {i + 1}: STOP')  # Etichetta per lo stato del generatore

            layout.addWidget(label)

            generator.status_label = label


            frequency_slider = QSlider(Qt.Horizontal)  # Slider per la frequenza

            frequency_slider.setMinimum(20)

            frequency_slider.setMaximum(2000)

            frequency_slider.valueChanged.connect(lambda val, idx=i: self.update_frequency(idx, val))

            layout.addWidget(frequency_slider)


            volume_slider = QSlider(Qt.Horizontal)  # Slider per il volume

            volume_slider.setMinimum(0)

            volume_slider.setMaximum(10)

            volume_slider.valueChanged.connect(lambda val, idx=i:
```

```
self.update_volume(idx, val))

        layout.addWidget(volume_slider)



        waveform_combo = QComboBox()  # ComboBox per selezionare la forma d'onda

        waveform_combo.addItems(['Sine', 'Square', 'Triangle', 'Sawtooth', 'Random'])

        waveform_combo.currentIndexChanged.connect(lambda combo_idx, idx=i: self.change_waveform(idx, combo_idx))

        layout.addWidget(waveform_combo)



        start_button = QPushButton('Start')  # Pulsante per avviare la riproduzione

        start_button.clicked.connect(lambda state, idx=i: self.start_audio_generation(idx))

        layout.addWidget(start_button)



        stop_button = QPushButton('Stop')  # Pulsante per fermare la riproduzione

        stop_button.clicked.connect(lambda state, idx=i: self.stop_audio_generation(idx))

        layout.addWidget(stop_button)



        reset_button = QPushButton('Reset')  # Pulsante per ripristinare le impostazioni predefinite

        reset_button.clicked.connect(lambda state, idx=i: self.reset_settings(idx))

        layout.addWidget(reset_button)
```

```
        save_button = QPushButton('Save')  # Pulsante per salvare l'audio generato
        save_button.clicked.connect(lambda state, idx=i: self.save_audio(idx))
        layout.addWidget(save_button)

    self.show()

# Avvia la riproduzione dell'audio per un generatore specifico
def start_audio_generation(self, idx):
    generator = self.generators[idx]
    generator.start()
    generator.status_label.setText(f'Generatore {idx + 1}: IS START')

# Ferma la riproduzione dell'audio per un generatore specifico
def stop_audio_generation(self, idx):
    generator = self.generators[idx]
    generator.stop()
    generator.status_label.setText(f'Generatore {idx + 1}: STOP')

# Ripristina le impostazioni predefinite per un generatore specifico
def reset_settings(self, idx):
    generator = self.generators[idx]
```

```
    generator.frequency = 440.0
    generator.volume = 0.5
    generator.waveform = 'Sine'
    generator.status_label.setText(f'Generatore {idx + 1}: STOP')

# Salva l'audio generato in un file WAV per un generatore specifico
def save_audio(self, idx):
    generator = self.generators[idx]
    wf = wave.open(f'audio_{idx}.wav', 'wb')
    wf.setnchannels(1)
    wf.setsampwidth(2)
    wf.setframerate(44100)
    wf.writeframes(generator.generate_audio(3))
    wf.close()

# Aggiorna la frequenza per un generatore specifico
def update_frequency(self, idx, value):
    generator = self.generators[idx]
    generator.frequency = value

# Aggiorna il volume per un generatore specifico
def update_volume(self, idx, value):
```

```
        generator = self.generators[idx]
        generator.volume = value / 10

    # Cambia la forma d'onda per un generatore specifico
    def change_waveform(self, idx, combo_idx):
        generator = self.generators[idx]
        if combo_idx == 0:
            generator.waveform = 'Sine'
        elif combo_idx == 1:
            generator.waveform = 'Square'
        elif combo_idx == 2:
            generator.waveform = 'Triangle'
        elif combo_idx == 3:
            generator.waveform = 'Sawtooth'
        else:
            generator.waveform = 'Random'

# Funzione principale per avviare l'applicazione
def main():
    app = QApplication(sys.argv)
    main_win = AudioGeneratorApp()
    sys.exit(app.exec_())
```

```
# Punto di ingresso dell'app

if __name__ == '__main__':

    main()
```

Archive USB Format

Archive USB Format Free è un'applicazione open-source sviluppata da Luca Bocaletto. Questo software offre una soluzione semplice ed efficiente per formattare le unità USB collegate al computer. Gli utenti possono selezionare il sistema di file desiderato (FAT32 o NTFS) e l'unità USB da formattare. Inoltre, l'applicazione rileva automaticamente le unità USB collegate, semplificando il processo di selezione. L'interfaccia utente intuitiva rende l'applicazione accessibile anche agli utenti meno esperti. Archive USB Format Free è distribuito sotto la licenza GPLv3, garantendo la libertà e la condivisione del software. Semplice da usare e altamente personalizzabile, questo software è un'ottima risorsa per la gestione delle unità USB e la formattazione rapida e affidabile.

Caratteristiche Principali

- Formattazione rapida delle unità USB
- Selezione del sistema di file (FAT32 o NTFS)
- Rilevamento automatico delle unità USB collegate
- Interfaccia utente intuitiva
- Distribuito con licenza GPLv3 per la condivisione e la personalizzazione
- Sviluppato da Luca Bocaletto

Source Code – Archive USB Format

```
# Software Name: Archive USB Format Free

# License: GPLv3

# Author: Bocaletto Luca



# Nome del Software: Archive USB Format Free
```

```
# Licenza: GPLv3
# Autore: Bocaletto Luca

# Importa le librerie necessarie
import tkinter as tk
import subprocess
import psutil

# Funzione per ottenere la lista delle unità USB collegate al computer
def get_usb_drives():
    usb_drives = []
    for partition in psutil.disk_partitions():
        if "removable" in partition.opts:
            usb_drives.append(partition.device)
    return usb_drives

# Funzione per formattare l'unità USB selezionata
def format_usb():
    selected_filesystem = filesystem_var.get()
    selected_drive = drive_var.get()

    # Verifica se è stata selezionata un'unità USB
```

```
    if not selected_drive:
        result_label.config(text="Seleziona una chiavetta USB.")
        return

    # Genera il comando di formattazione in base al sistema di file selezionato
    if selected_filesystem == "FAT32":
        cmd = f"format {selected_drive} /fs:FAT32 /q /y"
    elif selected_filesystem == "NTFS":
        cmd = f"format {selected_drive} /fs:NTFS /q /y"
    else:
        result_label.config(text="Seleziona un sistema di file valido.")
        return

    try:
        # Esegue il comando di formattazione
        subprocess.run(cmd, shell=True, check=True)
        result_label.config(text="Formattazione completata con successo.")
    except subprocess.CalledProcessError as e:
        result_label.config(text=f"Errore durante la formattazione: {e}")

# Crea la finestra principale dell'applicazione
root = tk.Tk()
```

```
root.title("Formatta Chiavetta USB")

# Etichetta per la selezione del sistema di file
filesystem_label = tk.Label(root, text="Seleziona il sistema di file:")
filesystem_label.pack()

# Variabile per la selezione del sistema di file
filesystem_var = tk.StringVar()
filesystem_var.set("FAT32")

# Menu a tendina per la selezione del sistema di file
filesystem_option_menu = tk.OptionMenu(root, filesystem_var, "FAT32", "NTFS")
filesystem_option_menu.pack()

# Etichetta per la selezione dell'unità USB
drive_label = tk.Label(root, text="Seleziona l'unità USB:")
drive_label.pack()

# Variabile per la selezione dell'unità USB
drive_var = tk.StringVar()

# Ottiene la lista delle unità USB collegate
```

```
drive_options = get_usb_drives()


# Verifica se sono presenti unità USB

if not drive_options:

    result_label = tk.Label(root, text="Nessuna chiavetta USB trovata. Collega una chiavetta USB e riprova.")

    result_label.pack()

else:

    # Menu a tendina per la selezione dell'unità USB

    drive_menu = tk.OptionMenu(root, drive_var, *drive_options)

    drive_menu.pack()


# Etichetta per il risultato dell'operazione di formattazione

result_label = tk.Label(root, text="")

result_label.pack()


# Pulsante per avviare la formattazione

format_button = tk.Button(root, text="Formatta", command=format_usb)

format_button.pack()


# Esegui l'applicazione

root.mainloop()
```

Calcolatrice Free

Calc Free è un'applicazione di calcolatrice semplice ma potente scritta in Python utilizzando la libreria PyQt6 per l'interfaccia utente. Questa applicazione consente agli utenti di eseguire operazioni matematiche di base, calcolare percentuali e calcolare radici quadrate e cubiche in modo intuitivo e facile da usare.

Funzionalità

- **Interfaccia Utente User-Friendly**: Calc Free offre un'interfaccia grafica intuitiva e facile da usare che consente agli utenti di inserire numeri e eseguire calcoli in modo agevole.
- **Operazioni Matematiche di Base**: Gli utenti possono eseguire operazioni di addizione (+), sottrazione (-), moltiplicazione (*) e divisione (/) con facilità.
- **Radici Quadrate e Cubiche**: Calc Free consente di calcolare sia la radice quadrata (sqrt) che la radice cubica (cbrt) di un numero.
- **Calcolo delle Percentuali**: Con l'operatore '%' gli utenti possono calcolare facilmente le percentuali.
- **Cancellazione dell'Input**: Il pulsante 'CANC' consente agli utenti di cancellare l'input corrente e iniziare un nuovo calcolo.

Come Usare

1. **Inserimento Numerico**: Usa i pulsanti numerici (0-9) e il punto decimale per inserire numeri. Il tuo input verrà visualizzato nella casella di testo superiore.
2. **Operazioni Matematiche**: Usa i pulsanti delle operazioni (+, -, *, /) per eseguire calcoli matematici di base. Il risultato verrà visualizzato nella casella di testo.
3. **Radici Quadrate e Cubiche**: Premi il pulsante 'sqrt' per calcolare la radice quadrata o il pulsante 'cbrt' per calcolare la radice cubica dell'input corrente.
4. **Calcolo delle Percentuali**: Usa il pulsante '%' per calcolare la percentuale basata sull'input corrente.
5. **Cancellazione dell'Input**: Clicca sul pulsante 'CANC' per cancellare l'input corrente e iniziare un nuovo calcolo.
6. **Calcolo Finale**: Premi il pulsante '=' per calcolare il risultato finale basato sull'input e sull'operatore corrente.

Source Code – Calcolatrice Free

```
# Software Name: Calc Free

# Author: Bocaletto Luca

import sys

from PyQt6.QtCore import Qt

from PyQt6.QtGui import QPalette, QColor, QAction

from PyQt6.QtWidgets import QApplication, QMainWindow, QWidget, QVBoxLayout, QHBoxLayout, QPushButton, QLineEdit, QMenuBar, QLabel, QDialog



class Calcolatrice(QMainWindow):

    def __init__(self):

        super().__init__()

        self.setWindowTitle("Calc Free")  # Imposta il titolo della finestra

        self.setGeometry(100, 100, 400, 400)  # Imposta le dimensioni e la posizione iniziale della finestra

        self.central_widget = QWidget()

        self.setCentralWidget(self.central_widget)  # Imposta il widget centrale come contenitore principale

        self.layout = QVBoxLayout()  # Crea un layout verticale per organizzare gli elementi

        self.display = QLineEdit()  # Crea una casella di testo per il display della calcolatrice

        self.display.setStyleSheet("font-size: 20px; background-color: #f0f0f0;")  # Stile per il display
```

```
        self.layout.addWidget(self.display)  # Aggiunge il display al layout

        button_layout = [  # Definizione dei pulsanti in un elenco di liste
            ["7", "8", "9", "+"],
            ["4", "5", "6", "-"],
            ["1", "2", "3", "*"],
            ["0", ".", "=", "/", "CANC"],
            ["sqrt", "cbrt", "%"]
        ]

        for row in button_layout:  # Itera sulle righe dei pulsanti
            row_layout = QHBoxLayout()  # Crea un layout orizzontale per organizzare i pulsanti in una riga
            for button_text in row:  # Itera sui testi dei pulsanti nella riga corrente
                button = QPushButton(button_text)  # Crea un pulsante con il testo corrente
                if button_text == "CANC":
                    button.setStyleSheet("font-size: 20px; background-color: #f44336; color: #fff;")  # Imposta lo stile per il pulsante CANCEL
                else:
                    button.setStyleSheet("font-size: 20px; background-color: #4caf50; color: #fff;")  # Imposta lo stile per gli altri pulsanti
                button.clicked.connect(self.button_click)  # Collega il click del pulsante alla funzione button_click
                row_layout.addWidget(button)  # Aggiunge il pulsante al layout della riga corrente
            self.layout.addLayout(row_layout)  # Aggiunge il layout della riga al layout principale
```

```
        self.central_widget.setLayout(self.layout)  # Imposta il layout principale per il widget centrale

        self.result = None  # Variabile per memorizzare il risultato

        self.operator = None  # Variabile per memorizzare l'operatore corrente

        self.clear_flag = False  # Flag per determinare se cancellare l'input

        self.current_input = ""  # Memorizza l'input dell'utente corrente

        self.percent_flag = False  # Flag per gestire il calcolo percentuale

        menubar = self.menuBar()  # Crea una barra del menu

        about_menu = menubar.addMenu("About")  # Crea un menu "About"

        about_action = QAction("About", self)  # Crea un'azione "About"

        about_menu.addAction(about_action)  # Aggiunge l'azione al menu "About"

        about_action.triggered.connect(self.show_about_dialog)  # Collega l'azione al metodo show_about_dialog


    def button_click(self):

        button = self.sender()  # Ottiene il pulsante che ha generato il segnale

        text = button.text()  # Ottiene il testo del pulsante premuto

        if text.isnumeric() or text == ".":  # Se il testo è numerico o un punto decimale

            self.current_input += text  # Aggiunge il testo all'input corrente

            self.display.setText(self.current_input)  # Aggiorna il display con l'input corrente

        elif text in ["+", "-", "*", "/"]:  # Se il testo è un operatore

            if self.result is None:  # Se non c'è ancora un risultato parziale

                self.result = float(self.current_input)  # Memorizza l'input corrente come
```

```
risultato
            else:
                self.result = self.perform_calculation()  # Altrimenti, esegui un calcolo parziale
            self.operator = text  # Memorizza l'operatore corrente
            self.clear_flag = True  # Imposta il flag per cancellare l'input
            self.current_input = ""  # Resetta l'input corrente
            self.percent_flag = False  # Resetta il flag percentuale
        elif text == "=":  # Se il testo è "=", esegui il calcolo finale
            if self.operator:
                if self.current_input:
                    operand = float(self.current_input)
                    if self.operator == "%":  # Se l'operatore è "%", calcola la percentuale
                        result = self.result * (operand / 100)
                    else:
                        result = self.perform_calculation(operand)  # Altrimenti, esegui l'operazione corrispondente
                    self.display.setText(str(result))  # Mostra il risultato nel display
                    self.clear_flag = True  # Imposta il flag per cancellare l'input successivo
                    self.operator = None  # Resetta l'operatore
                    self.current_input = ""  # Resetta l'input corrente
        elif text == "sqrt":  # Se il testo è "sqrt", calcola la radice quadrata
            if self.current_input:
```

```
            num = float(self.current_input)
            if num >= 0:
                result = num ** 0.5
                self.display.setText(str(result))
                self.current_input = str(result)
                self.percent_flag = False  # Resetta il flag percentuale
        elif text == "cbrt":  # Se il testo è "cbrt", calcola la radice cubica
            if self.current_input:
                num = float(self.current_input)
                result = num ** (1/3)
                self.display.setText(str(result))
                self.current_input = str(result)
                self.percent_flag = False  # Resetta il flag percentuale
        elif text == "%":  # Se il testo è "%", gestisci il calcolo percentuale
            if self.current_input and not self.percent_flag:
                value = float(self.current_input)
                self.result = value  # Memorizza il valore come risultato parziale
                self.operator = "%"  # Imposta l'operatore come "%"
                self.display.setText(self.current_input + "%")  # Mostra il simbolo "%" nel
display
                self.current_input = ""  # Resetta l'input corrente
                self.percent_flag = True  # Imposta il flag percentuale
        elif text == "CANC":  # Se il testo è "CANC", resetta tutto
```

```
        self.current_input = ""
        self.display.clear()
        self.result = None

def perform_calculation(self, operand=None):
    if operand is None:
        operand = float(self.current_input)
    if self.operator == "+":
        return self.result + operand
    elif self.operator == "-":
        return self.result - operand
    elif self.operator == "*":
        return self.result * operand
    elif self.operator == "/":
        if operand != 0:
            return self.result / operand
        else:
            return "Errore"

def show_about_dialog(self):
    about_dialog = QDialog(self)  # Crea una finestra di dialogo "About"
    about_dialog.setWindowTitle("About")  # Imposta il titolo della finestra
```

```
        about_dialog.setFixedWidth(300)  # Imposta la larghezza fissa della finestra

        about_dialog.setFixedHeight(150)  # Imposta l'altezza fissa della finestra



        about_layout = QVBoxLayout()  # Crea un layout verticale per la finestra di dialogo

        about_label = QLabel("Questa è una semplice calcolatrice GUI in Python con PyQt6.")  # Crea una label con testo



        # Imposta il colore del testo su bianco

        about_label.setStyleSheet("color: #fff;")



        # Imposta lo sfondo della finestra "About" come i pulsanti

        palette = about_dialog.palette()

        palette.setColor(QPalette.ColorRole.Window, QColor("#4caf50"))  # Colore personalizzato

        about_dialog.setPalette(palette)



        about_layout.addWidget(about_label)  # Aggiunge la label al layout

        about_dialog.setLayout(about_layout)  # Imposta il layout per la finestra di dialogo



        about_dialog.exec()  # Esegui la finestra di dialogo



if __name__ == "__main__":
```

```
    app = QApplication(sys.argv)  # Crea un'applicazione PyQt

    window = Calcolatrice()  # Crea l'istanza della Calcolatrice

    about_action = window.menuBar().findChild(QAction, "About")  # Trova l'azione "About" nel menu

    if about_action:

        about_action.setStyleSheet("color: #fff;")  # Imposta il colore del testo dell'azione "About" in bianco



    window.show()  # Mostra la finestra della Calcolatrice

    sys.exit(app.exec())  # Esegui l'applicazione PyQt
```

SD-Card Format

Sd Card Format Free è un'applicazione software sviluppata da Bocaletto Luca per semplificare il processo di formattazione delle schede di memoria SD. Questo software offre un'interfaccia utente intuitiva e facile da usare che consente agli utenti di formattare rapidamente le loro schede SD in formato FAT32.

Caratteristiche principali

- **Selezione dell'Unità Rimovibile:** SD-Card Format Free permette agli utenti di selezionare l'unità rimovibile desiderata da una comoda lista a discesa. Questo è particolarmente utile quando si hanno più schede di memoria SD collegate al computer.

- **Aggiornamento della Lista delle Unità Rimovibili:** L'applicazione include un pulsante "Aggiorna" che consente agli utenti di rilevare automaticamente le unità rimovibili attualmente collegate al computer. Questo assicura che la lista sia sempre aggiornata e che gli utenti possano selezionare facilmente l'unità che desiderano formattare.

- **Formattazione Facile:** Una volta selezionata l'unità desiderata, gli utenti possono avviare il processo di formattazione con un semplice clic sul pulsante "Formatta". Il software eseguirà quindi il comando di formattazione con il file system FAT32 sull'unità selezionata.

- **Notifiche di Stato:** SD-Card Format Free fornisce notifiche informative all'utente. Una

finestra di messaggio mostra un messaggio di conferma quando la formattazione è completata con successo. In caso di errore durante il processo, viene visualizzato un messaggio di errore per informare l'utente sul problema riscontrato.

Source Code – SD Card Format

```
# Software Name: SD-Card Format Free
# Author: Bocaletto Luca

import tkinter as tk

from tkinter import ttk

import subprocess

import ctypes

import psutil

# Funzione per ottenere una lista delle unità rimovibili

def get_removable_drives():

    drives = []

    for drive in psutil.disk_partitions():

        if "removable" in drive.opts:

            drives.append(drive.device)

    return drives

# Funzione per formattare la scheda di memoria SD

def formatta_scheda():

    dispositivo = dispositivo_combobox.get()

    try:

        comando = f"format {dispositivo} /FS:FAT32"
```

```
        subprocess.run(comando, shell=True, check=True)

        messagebox.showinfo("Formattazione Completata", f"La scheda {dispositivo} è stata formattata con successo.")

    except Exception as e:

        ctypes.windll.user32.MessageBoxW(0, f"Impossibile formattare la scheda {dispositivo}.\nErrore: {str(e)}", "Errore", 0)

# Funzione per aggiornare la lista delle unità rimovibili

def refresh_drives():

    dispositivi = get_removable_drives()

    dispositivo_combobox['values'] = dispositivi

# Creazione della finestra principale

root = tk.Tk()

root.title("Applicativo per Formattare Schede di Memoria SD")

# Etichetta e combobox per selezionare l'unità

dispositivo_label_title = tk.Label(root, text="SD-Card Format Free")

dispositivo_label = tk.Label(root, text="Seleziona un'Unità Rimovibile:")

dispositivo_label_title.pack()

dispositivo_label.pack()

dispositivo_combobox = ttk.Combobox(root, state="readonly")

dispositivo_combobox.pack()

# Pulsante per aggiornare la lista delle unità rimovibili

aggiorna_button = tk.Button(root, text="Aggiorna", command=refresh_drives)

aggiorna_button.pack()
```

```
# Pulsante per avviare la formattazione

formatta_button = tk.Button(root, text="Formatta", command=formatta_scheda)

formatta_button.pack()

# Esegui l'aggiornamento iniziale della lista delle unità rimovibili

refresh_drives()

# Esecuzione del loop principale della GUI

root.mainloop()
```

Event Calendar Free

Event Calendar Free è un'applicazione per la gestione degli eventi basata su una GUI (Interfaccia Grafica Utente) sviluppata utilizzando il framework PyQt5. Questa applicazione consente agli utenti di visualizzare, aggiungere ed eliminare eventi in un calendario interattivo.

Caratteristiche principali

1. **Calendario Interattivo**: L'applicazione presenta un calendario interattivo che consente agli utenti di selezionare una data specifica facendo clic su di essa.

2. **Visualizzazione degli Eventi**: Quando un utente seleziona una data nel calendario, l'applicazione mostra una finestra dedicata denominata "Eventi del [Data selezionata]" che visualizza gli eventi associati a quella data.

3. **Aggiunta di Eventi**: Gli utenti possono aggiungere nuovi eventi inserendo un nome e una descrizione per l'evento e facendo clic sul pulsante "Aggiungi Evento". Questi eventi vengono memorizzati in un database SQLite.

4. **Eliminazione di Eventi**: Gli utenti possono eliminare eventi selezionando le righe corrispondenti nella tabella degli eventi e facendo clic sul pulsante "Elimina Evento". Gli eventi eliminati vengono rimossi dal database SQLite.

5. **Visualizzazione Dettagli Evento**: Gli eventi vengono visualizzati in una tabella che mostra l'ID dell'evento, il nome e la descrizione. Questi dettagli aiutano gli utenti a identificare e gestire gli eventi.

6. **Persistenza dei Dati**: I dati sugli eventi vengono memorizzati in un database SQLite chiamato 'events.db', consentendo agli utenti di accedere agli eventi anche dopo la chiusura e

la riapertura dell'applicazione.

7. **Validazione degli Input**: L'applicazione include una validazione per assicurarsi che l'utente inserisca almeno il nome dell'evento prima di aggiungerlo.

Utilizzo

1. All'avvio, l'applicazione mostra un calendario interattivo.
2. Gli utenti possono selezionare una data specifica facendo clic su di essa nel calendario.
3. Quando viene selezionata una data, viene visualizzata una finestra separata che mostra gli eventi associati a quella data.
4. Gli utenti possono aggiungere nuovi eventi inserendo il nome e la descrizione dell'evento e facendo clic su "Aggiungi Evento".
5. Gli utenti possono eliminare eventi selezionando le righe corrispondenti nella tabella degli eventi e facendo clic su "Elimina Evento".
6. Tutti gli eventi vengono memorizzati in modo persistente nel database 'events.db', quindi possono essere consultati in futuro.

Source Code – Event Calendar Free

Galleria di Immagini in Python con GUI Qt6

Autore: Bocaletto Luca

La "Galleria di Immagini in Python con GUI Qt6" è un'applicazione che ti consente di visualizzare e sfogliare immagini da una directory specifica sul tuo computer.

Caratteristiche Principali

- **Interfaccia Utente Intuitiva**: La Galleria di Immagini offre un'interfaccia utente intuitiva e facile da usare, grazie alla potente libreria Qt6.

- **Visualizzazione di Immagini**: Puoi visualizzare immagini in diversi formati, tra cui JPG, JPEG, PNG, GIF e BMP, direttamente all'interno dell'applicazione.

- **Navigazione Facile**: L'applicazione consente di sfogliare facilmente le immagini nella directory selezionata, grazie a pulsanti "Avanti" e "Indietro" che ti permettono di passare da un'immagine all'altra.

- **Zoom Personalizzabile**: Puoi ingrandire o rimpicciolire l'immagine attualmente visualizzata usando la barra per lo zoom.

- **Rotazione Automatica**: L'applicazione è in grado di rilevare l'orientamento Exif delle immagini e ruotarle automaticamente per una visualizzazione corretta.

- **Miniature**: Una lista di miniature delle immagini presenti nella directory ti permette di selezionare rapidamente l'immagine desiderata.
- **Esplorazione delle Directory**: L'applicazione offre la possibilità di esplorare facilmente le directory sul tuo computer per trovare e visualizzare immagini.
- **Informazioni "About"**: Puoi accedere alle informazioni "About" dell'applicazione per scoprire la versione e l'autore.

Uso

Per utilizzare la Galleria di Immagini, avvia l'applicazione e seleziona una directory che contiene le immagini che desideri visualizzare. Puoi quindi utilizzare i pulsanti "Avanti" e "Indietro" per navigare tra le immagini, ingrandirle o rimpicciolirle tramite lo zoom, e fare clic sulle miniature nella lista per selezionare un'immagine specifica.

Source Code – Event Calendar Free

```
# Author: Bocaletto Luca
# Gallery Image in Python with GUI Qt6

# Galleria di Immagini in Python con GUI Qt6

# Import delle librerie necessarie

import sys

import os

from PySide6.QtWidgets import QApplication, QMainWindow, QPushButton, QVBoxLayout, QWidget, QFileDialog, QGraphicsView, QGraphicsScene, QGraphicsPixmapItem, QSlider, QListWidget, QListWidgetItem, QHBoxLayout, QMessageBox

from PySide6.QtGui import QPixmap, QImage, Qt, QTransform, QIcon

from PIL import Image

from pathlib import Path

from PySide6.QtCore import QSize
```

```
# Definizione della classe principale dell'applicazione
class ImageGallery(QMainWindow):
    def __init__(self, default_directory):
        super().__init__()

        # Impostazioni della finestra principale
        self.setWindowTitle("Galleria di Immagini")
        self.setGeometry(100, 100, 1024, 768)
        # Inizializzazione delle variabili
        self.image_list = []
        self.current_index = 0
        # Creazione del widget centrale
        self.central_widget = QWidget()
        self.setCentralWidget(self.central_widget)
        # Layout principale
        self.layout = QVBoxLayout(self.central_widget)
        # Elemento grafico per l'immagine principale
        self.image_label = QGraphicsPixmapItem()

        # Vista grafica per l'immagine principale
        self.graphics_view = QGraphicsView(self)
        self.layout.addWidget(self.graphics_view)
```

```
# Scene per l'immagine principale
self.scene = QGraphicsScene()
self.graphics_view.setScene(self.scene)
self.scene.addItem(self.image_label)
# Layout orizzontale per i pulsanti
button_layout = QHBoxLayout()
# Pulsanti per l'immagine
self.previous_button = QPushButton("Indietro", self)
self.previous_button.clicked.connect(self.show_previous_image)
button_layout.addWidget(self.previous_button)
self.next_button = QPushButton("Avanti", self)
self.next_button.clicked.connect(self.show_next_image)
button_layout.addWidget(self.next_button)
self.browse_button = QPushButton("Sfoglia", self)
self.browse_button.clicked.connect(self.browse_images)
button_layout.addWidget(self.browse_button)
# Pulsante "About"
self.about_button = QPushButton("About", self)
self.about_button.clicked.connect(self.show_about_dialog)
button_layout.addWidget(self.about_button)
# Barra per lo zoom
self.zoom_slider = QSlider(Qt.Horizontal)
```

```
        self.zoom_slider.setMinimum(10)
        self.zoom_slider.setMaximum(400)
        self.zoom_slider.setValue(100)
        self.zoom_slider.valueChanged.connect(self.zoom_image)
        button_layout.addWidget(self.zoom_slider)
        self.layout.addLayout(button_layout)
        # Slide show con miniature
        self.thumbnail_list = QListWidget(self)
        self.thumbnail_list.setMaximumHeight(220)
        self.thumbnail_list.setViewMode(QListWidget.IconMode)
        self.thumbnail_list.setIconSize(QSize(200, 200))
        self.thumbnail_list.setGridSize(QSize(220, 220))
        self.thumbnail_list.setLayoutMode(QListWidget.Batched)
        self.thumbnail_list.itemClicked.connect(self.thumbnail_clicked)
        self.layout.addWidget(self.thumbnail_list)
        # Utilizza la directory home come directory di default
        self.load_images_from_directory(default_directory)
    # Funzione per sfogliare le immagini in una directory
    def browse_images(self):
        options = QFileDialog.Options()
        options |= QFileDialog.ReadOnly
        dialog = QFileDialog(self, "Scegli una directory con immagini",
options=options)
```

```
        dialog.setFileMode(QFileDialog.Directory)
        dialog.setOption(QFileDialog.ShowDirsOnly, True)
        if dialog.exec():
            directory = dialog.selectedFiles()[0]
            self.load_images_from_directory(directory)

    # Funzione per caricare le immagini da una directory
    def load_images_from_directory(self, directory):
        self.image_list = [os.path.join(directory, filename) for filename in
os.listdir(directory) if
                           filename.lower().endswith(('.jpg', '.jpeg', '.png', '.gif', '.bmp'))]
        # Pulisci la lista delle miniature
        self.thumbnail_list.clear()
        if self.image_list:
            self.current_index = 0
            self.show_image(self.current_index)
            # Aggiungi le miniature alla lista
            for image_path in self.image_list:
                image = Image.open(image_path)
                image = self.rotate_image(image)  # Ruota l'immagine se necessario
                # Crea la miniatura a partire dall'immagine ruotata
                thumbnail = image.copy()
                thumbnail.thumbnail((200, 200), Image.LANCZOS)  # Utilizza LANCZOS
```

per l'antialiasing

```
            # Converte la miniatura in un formato compatibile con QPixmap
            thumbnail_qimage = self.pillow_to_qimage(thumbnail)
            thumbnail_pixmap = QPixmap.fromImage(thumbnail_qimage)
            item = QListWidgetItem(QIcon(thumbnail_pixmap), "")
            self.thumbnail_list.addItem(item)
        else:
            self.image_label.setPixmap(QPixmap())
    # Funzione per mostrare un'immagine in base all'indice
    def show_image(self, index):
        if 0 <= index < len(self.image_list):
            image_path = self.image_list[index]
            image = Image.open(image_path)
            image = self.rotate_image(image)  # Ruota l'immagine se necessario
            qimage = self.pillow_to_qimage(image)
            pixmap = QPixmap.fromImage(qimage)
            self.image_label.setPixmap(pixmap)
            self.scene.setSceneRect(0, 0, pixmap.width(), pixmap.height())
            self.graphics_view.setScene(self.scene)
            self.graphics_view.fitInView(self.scene.sceneRect(), Qt.KeepAspectRatio)
    # Funzione per ruotare un'immagine in base all'orientamento Exif
```

```
def rotate_image(self, image):
    try:
        exif = image._getexif()
        if exif:
            orientation = exif.get(0x0112)
            if orientation == 3:
                image = image.rotate(180, expand=True)
            elif orientation == 6:
                image = image.rotate(270, expand=True)
            elif orientation == 8:
                image = image.rotate(90, expand=True)
    except (AttributeError, KeyError, IndexError):
        pass
    return image
# Funzione per convertire un'immagine da formato Pillow a QImage
def pillow_to_qimage(self, image):
    if image.mode == "RGB":
        image = image.convert("RGBA")
    width, height = image.size
    image_data = image.tobytes("raw", "RGBA")
    return QImage(image_data, width, height, QImage.Format_RGBA8888)
# Funzione per mostrare l'immagine precedente
```

```
    def show_previous_image(self):
        if self.current_index > 0:
            self.current_index -= 1
            self.show_image(self.current_index)
    # Funzione per mostrare l'immagine successiva
    def show_next_image(self):
        if self.current_index < len(self.image_list) - 1:
            self.current_index += 1
            self.show_image(self.current_index)
    # Funzione per gestire lo zoom dell'immagine
    def zoom_image(self):
        zoom_factor = self.zoom_slider.value() / 100.0
        self.graphics_view.setTransform(QTransform.fromScale(zoom_factor,
zoom_factor))
    # Funzione per gestire il click su una miniatura
    def thumbnail_clicked(self, item):
        index = self.thumbnail_list.indexFromItem(item).row()
        self.show_image(index)
    # Funzione per mostrare il dialog "About"
    def show_about_dialog(self):
        about_text = "Galleria di Immagini Versione 1.0\n\n" \
                     "Applicazione per visualizzare e sfogliare immagini da una directory.\n"
\
```

```
                "Realizzata con Python con GUI Qt6.\n\n" \
                "2023 Bocaletto Luca Aka Elektronoide"
        QMessageBox.about(self, "About", about_text)
# Blocco principale di esecuzione dell'applicazione
if __name__ == "__main__":
    app = QApplication(sys.argv)

    # Utilizza la directory home come directory di default
    default_directory = str(Path.home())
    gallery = ImageGallery(default_directory)
    gallery.show()
    sys.exit(app.exec())
```

Mind Maps Free

"Mappe Mentali" è un'applicazione software per la creazione, gestione e organizzazione di mappe mentali. Questo strumento permette agli utenti di creare mappe mentali personalizzate, organizzare idee e concetti in modo gerarchico, e navigare attraverso le informazioni in modo intuitivo.

Funzionalità Principali

- **Creazione di Mappe Mentali:** Gli utenti possono facilmente creare nuove mappe mentali specificando il nome e, se necessario, una descrizione.
- **Gestione delle Mappe:** Le mappe mentali esistenti possono essere visualizzate, modificate e cancellate direttamente dall'interfaccia dell'applicazione.
- **Creazione di Nodi Padre:** Per ogni mappa mentale, è possibile inserire nodi padre, che rappresentano i principali concetti o argomenti. Ogni nodo padre può essere descritto con un nome e una descrizione.
- **Creazione di Nodi Figlio:** Per espandere ulteriormente le informazioni, gli utenti possono aggiungere nodi figlio a ciascun nodo padre. Questi nodi figlio sono collegati

gerarchicamente e possono contenere nome e descrizione.

- **Navigazione Intuitiva:** La navigazione all'interno delle mappe mentali è intuitiva: gli utenti possono espandere o contrarre i nodi figlio collegati ai nodi padre per esplorare le informazioni.
- **Eliminazione:** Le mappe mentali, i nodi padre e i nodi figlio possono essere cancellati quando non sono più necessari.
- **Interfaccia Utente User-Friendly:** L'interfaccia dell'applicazione è progettata per essere intuitiva e facile da usare, permettendo agli utenti di concentrarsi sulla creazione e l'organizzazione delle mappe mentali.
- **Archiviazione Sicura:** Tutte le mappe mentali e i dati correlati sono archiviati in modo sicuro nel database dell'applicazione.

Source Code - Mind Maps Free

```
# Name Software: Mind Maps Free
# Author: Bocaletto Luca
# Importazione delle librerie necessarie

import sys

from PyQt5.QtWidgets import QApplication, QMainWindow, QWidget, QVBoxLayout, QHBoxLayout, QLabel, QLineEdit, QPushButton, QTableWidget, QTableWidgetItem, QHeaderView, QDialog, QMessageBox

import sqlite3

# Definizione della classe principale dell'applicazione

class MappeMentaliApp(QMainWindow):

    def __init__(self):

        super().__init__()

        self.setWindowTitle("Software Mappe Mentali")

        self.setGeometry(100, 100, 800, 600)

        # Creazione del widget centrale
```

```
        self.central_widget = QWidget()
        self.setCentralWidget(self.central_widget)
        # Inizializzazione dell'interfaccia utente
        self.init_ui()
    def init_ui(self):
        # Creazione del layout principale
        layout = QVBoxLayout()
        # Layout per l'inserimento delle mappe mentali
        inserimento_layout = QHBoxLayout()
        self.nome_mappa_input = QLineEdit()
        self.descrizione_mappa_input = QLineEdit()
        btn_inserisci = QPushButton("Inserisci")
        btn_inserisci.clicked.connect(self.inserisci_mappa)
        inserimento_layout.addWidget(QLabel("Nome Mappa:"))
        inserimento_layout.addWidget(self.nome_mappa_input)
        inserimento_layout.addWidget(QLabel("Descrizione Mappa:"))
        inserimento_layout.addWidget(self.descrizione_mappa_input)
        inserimento_layout.addWidget(btn_inserisci)
        # Tabella per visualizzare le mappe mentali
        self.tabella_mappe = QTableWidget()
        self.tabella_mappe.setColumnCount(3)
        self.tabella_mappe.setHorizontalHeaderLabels(["ID", "Nome", "Descrizione"])
```

```
self.tabella_mappe.horizontalHeader().setSectionResizeMode(QHeaderView.Stretch)
        self.tabella_mappe.cellClicked.connect(self.apri_nodi_padre)
        self.carica_mappe()
        # Pulsante per eliminare mappe mentali
        btn_elimina = QPushButton("Elimina")
        btn_elimina.clicked.connect(self.elimina_mappa)
        # Aggiunta dei widget al layout principale
        layout.addLayout(inserimento_layout)
        layout.addWidget(self.tabella_mappe)
        layout.addWidget(btn_elimina)
        container = QWidget()
        container.setLayout(layout)
        self.setCentralWidget(container)
        self.finestra_nodi_padre = FinestraNodiPadre()
    # Connessione al database o creazione se non esiste
    def connetti_o_crea_database(self):
        conn = sqlite3.connect("DATABASE.db")
        conn.execute('''CREATE TABLE IF NOT EXISTS mappe_mentali
                        (id INTEGER PRIMARY KEY AUTOINCREMENT,
                        nome TEXT NOT NULL,
                        descrizione TEXT)''')
        # Crea la tabella nodi_padre se non esiste
```

```
    conn.execute('''CREATE TABLE IF NOT EXISTS nodi_padre
                    (id INTEGER PRIMARY KEY AUTOINCREMENT,
                    nome TEXT NOT NULL,
                    descrizione TEXT,
                    mappa_id INTEGER,
                    FOREIGN KEY (mappa_id) REFERENCES mappe_mentali(id))''')
    # Crea la tabella nodi_figlio se non esiste
    conn.execute('''CREATE TABLE IF NOT EXISTS nodi_figlio
                    (id INTEGER PRIMARY KEY AUTOINCREMENT,
                    nome TEXT NOT NULL,
                    descrizione TEXT,
                    padre_id INTEGER,
                    FOREIGN KEY (padre_id) REFERENCES nodi_padre(id))''')
    conn.commit()
    return conn
# Caricamento delle mappe mentali nella tabella
def carica_mappe(self):
    self.tabella_mappe.setRowCount(0)
    conn = self.connetti_o_crea_database()
    if conn:
        cursor = conn.cursor()
        cursor.execute("SELECT id, nome, descrizione FROM mappe_mentali")
```

```
        for row_idx, row_data in enumerate(cursor.fetchall()):
            self.tabella_mappe.insertRow(row_idx)
            for col_idx, cell_data in enumerate(row_data):
                self.tabella_mappe.setItem(row_idx, col_idx,
QTableWidgetItem(str(cell_data)))
        conn.close()

    # Inserimento di una nuova mappa mentale
    def inserisci_mappa(self):
        nome_mappa = self.nome_mappa_input.text()
        descrizione_mappa = self.descrizione_mappa_input.text()
        conn = self.connetti_o_crea_database()
        if conn:
            cursor = conn.cursor()
            cursor.execute("INSERT INTO mappe_mentali (nome, descrizione)
VALUES (?, ?)", (nome_mappa, descrizione_mappa))
            conn.commit()
            conn.close()
            self.carica_mappe()
            self.nome_mappa_input.clear()
            self.descrizione_mappa_input.clear()
    # Eliminazione di una mappa mentale
    def elimina_mappa(self):
```

```
        selected_row = self.tabella_mappe.currentRow()
        if selected_row >= 0:
            mappa_id = int(self.tabella_mappe.item(selected_row, 0).text())
            conn = self.connetti_o_crea_database()
            if conn:
                cursor = conn.cursor()
                cursor.execute("DELETE FROM mappe_mentali WHERE id=?",
(mappa_id,))
                conn.commit()
                conn.close()
                self.carica_mappe()
    # Apertura della finestra dei nodi padre per una mappa mentale specifica
    def apri_nodi_padre(self, row, col):
        mappa_id = int(self.tabella_mappe.item(row, 0).text())
        nome_mappa = self.tabella_mappe.item(row, 1).text()
        self.finestra_nodi_padre.mostra_nodi_padre(mappa_id, nome_mappa)
# Definizione della classe per la finestra dei nodi padre
class FinestraNodiPadre(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Nodi Padre")
        self.setGeometry(200, 200, 400, 300)
        self.layout = QVBoxLayout()
```

```
        self.label_titolo = QLabel("Label Nodi Padre")
        self.layout.addWidget(self.label_titolo)

        # Elementi per l'inserimento/modifica/eliminazione del nodo padre
        self.nome_nodo_padre_input = QLineEdit()
        self.descrizione_nodo_padre_input = QLineEdit()
        self.btn_inserisci_nodo_padre = QPushButton("Inserisci Nodo Padre")
        self.btn_elimina_nodo_padre = QPushButton("Elimina Nodo Padre")
        self.btn_inserisci_nodo_padre.clicked.connect(self.inserisci_nodo_padre)
        self.btn_elimina_nodo_padre.clicked.connect(self.elimina_nodo_padre)
        # Tabella per visualizzare i nodi padre associati alla mappa mentale
        self.tabella_nodi_padre = QTableWidget()
        self.tabella_nodi_padre.setColumnCount(3)
        self.tabella_nodi_padre.setHorizontalHeaderLabels(["ID", "Nome",
"Descrizione"])

self.tabella_nodi_padre.horizontalHeader().setSectionResizeMode(QHeaderView.Str
etch)
        self.layout.addWidget(QLabel("Inserisci Nodo Padre:"))
        self.layout.addWidget(QLabel("Nome Nodo Padre:"))
        self.layout.addWidget(self.nome_nodo_padre_input)
        self.layout.addWidget(QLabel("Descrizione Nodo Padre:"))
        self.layout.addWidget(self.descrizione_nodo_padre_input)
        self.layout.addWidget(self.btn_inserisci_nodo_padre)
```

```
        self.layout.addWidget(self.btn_elimina_nodo_padre)
        self.layout.addWidget(QLabel("Nodi Padre associati alla Mappa Mentale:"))
        self.layout.addWidget(self.tabella_nodi_padre)
        self.tabella_nodi_padre.cellClicked.connect(self.apri_nodi_figlio)
        self.setLayout(self.layout)
    # Mostra i nodi padre associati a una mappa mentale specifica
    def mostra_nodi_padre(self, mappa_id, nome_mappa):
        self.mappa_id = mappa_id
        self.label_titolo.setText(f"Nodi Padre - Mappa: {nome_mappa} (ID: {mappa_id})")
        self.carica_nodi_padre()
        self.show()
    # Carica i nodi padre associati alla mappa mentale nella tabella
    def carica_nodi_padre(self):
        self.tabella_nodi_padre.setRowCount(0)
        conn = sqlite3.connect("DATABASE.db")
        if conn:
            cursor = conn.cursor()
            cursor.execute("SELECT id, nome, descrizione FROM nodi_padre WHERE mappa_id=?", (self.mappa_id,))
            for row_idx, row_data in enumerate(cursor.fetchall()):
                self.tabella_nodi_padre.insertRow(row_idx)
                for col_idx, cell_data in enumerate(row_data):
```

```
                self.tabella_nodi_padre.setItem(row_idx, col_idx,
QTableWidgetItem(str(cell_data)))

        conn.close()


    # Inserimento di un nuovo nodo padre associato a una mappa mentale

    def inserisci_nodo_padre(self):

        nome_nodo_padre = self.nome_nodo_padre_input.text()

        descrizione_nodo_padre = self.descrizione_nodo_padre_input.text()

        conn = sqlite3.connect("DATABASE.db")

        if conn:

            cursor = conn.cursor()

            cursor.execute("INSERT INTO nodi_padre (nome, descrizione, mappa_id)
VALUES (?, ?, ?)",

                           (nome_nodo_padre, descrizione_nodo_padre, self.mappa_id))

            conn.commit()

            conn.close()

            self.carica_nodi_padre()

            self.nome_nodo_padre_input.clear()

            self.descrizione_nodo_padre_input.clear()


    # Eliminazione di un nodo padre selezionato

    def elimina_nodo_padre(self):

        selected_row = self.tabella_nodi_padre.currentRow()
```

```
        if selected_row >= 0:
            nodo_padre_id = int(self.tabella_nodi_padre.item(selected_row, 0).text())
            conn = sqlite3.connect("DATABASE.db")
            if conn:
                cursor = conn.cursor()
                cursor.execute("DELETE FROM nodi_padre WHERE id=?",
(nodo_padre_id,))
                conn.commit()
                conn.close()
                self.carica_nodi_padre()
    # Apertura della finestra dei nodi figlio per un nodo padre specifico
    def apri_nodi_figlio(self, row, col):
        nodo_padre_id = int(self.tabella_nodi_padre.item(row, 0).text())
        nome_nodo_padre = self.tabella_nodi_padre.item(row, 1).text()
        finestra_nodi_figlio = FinestraNodiFiglio(nodo_padre_id, nome_nodo_padre)
        finestra_nodi_figlio.exec_()
# Definizione della classe per la finestra dei nodi figlio
class FinestraNodiFiglio(QDialog):
    def __init__(self, nodo_padre_id, nome_nodo_padre):
        super().__init__()
        self.setWindowTitle("Nodi Figlio")
        self.setGeometry(300, 300, 400, 300)
        self.layout = QVBoxLayout()
```

```
        self.label_titolo = QLabel(f"Nodi Figlio - Nodo Padre: {nome_nodo_padre} (ID: {nodo_padre_id})")

        self.layout.addWidget(self.label_titolo)

        self.nodo_padre_id = nodo_padre_id  # Memorizza l'ID del Nodo Padre

        # Elementi per l'inserimento/modifica/eliminazione del nodo figlio

        self.nome_nodo_figlio_input = QLineEdit()

        self.descrizione_nodo_figlio_input = QLineEdit()

        self.btn_inserisci_nodo_figlio = QPushButton("Inserisci Nodo Figlio")

        self.btn_elimina_nodo_figlio = QPushButton("Elimina Nodo Figlio")

        self.btn_inserisci_nodo_figlio.clicked.connect(self.inserisci_nodo_figlio)

        self.btn_elimina_nodo_figlio.clicked.connect(self.elimina_nodo_figlio)

        # Tabella per visualizzare i nodi figlio associati al nodo padre

        self.tabella_nodi_figlio = QTableWidget()

        self.tabella_nodi_figlio.setColumnCount(3)

        self.tabella_nodi_figlio.setHorizontalHeaderLabels(["ID", "Nome", "Descrizione"])
self.tabella_nodi_figlio.horizontalHeader().setSectionResizeMode(QHeaderView.Stretch)

        self.layout.addWidget(QLabel("Inserisci Nodo Figlio:"))

        self.layout.addWidget(QLabel("Nome Nodo Figlio:"))

        self.layout.addWidget(self.nome_nodo_figlio_input)

        self.layout.addWidget(QLabel("Descrizione Nodo Figlio:"))

        self.layout.addWidget(self.descrizione_nodo_figlio_input)

        self.layout.addWidget(self.btn_inserisci_nodo_figlio)
```

```
        self.layout.addWidget(self.btn_elimina_nodo_figlio)
        self.layout.addWidget(QLabel("Nodi Figlio associati al Nodo Padre:"))
        self.layout.addWidget(self.tabella_nodi_figlio)
        self.tabella_nodi_figlio.cellClicked.connect(self.seleziona_nodo_figlio)
        self.setLayout(self.layout)
        # Carica i dati dei nodi figlio all'apertura della finestra
        self.carica_nodi_figlio()
    # Carica i nodi figlio associati al nodo padre nella tabella
    def carica_nodi_figlio(self):
        self.tabella_nodi_figlio.setRowCount(0)
        conn = sqlite3.connect("DATABASE.db")
        if conn:
            cursor = conn.cursor()
            cursor.execute("SELECT id, nome, descrizione FROM nodi_figlio WHERE
padre_id=?", (self.nodo_padre_id,))
            for row_idx, row_data in enumerate(cursor.fetchall()):
                self.tabella_nodi_figlio.insertRow(row_idx)
                for col_idx, cell_data in enumerate(row_data):
                    self.tabella_nodi_figlio.setItem(row_idx, col_idx,
QTableWidgetItem(str(cell_data)))
            conn.close()
    # Inserimento di un nuovo nodo figlio associato a un nodo padre
    def inserisci_nodo_figlio(self):
```

```
        nome_nodo_figlio = self.nome_nodo_figlio_input.text()
        descrizione_nodo_figlio = self.descrizione_nodo_figlio_input.text()
        conn = sqlite3.connect("DATABASE.db")
        if conn:
            cursor = conn.cursor()
            cursor.execute("INSERT INTO nodi_figlio (nome, descrizione, padre_id)
VALUES (?, ?, ?)",
                           (nome_nodo_figlio, descrizione_nodo_figlio, self.nodo_padre_id))
            conn.commit()
            conn.close()
            self.carica_nodi_figlio()
            self.nome_nodo_figlio_input.clear()
            self.descrizione_nodo_figlio_input.clear()
    # Eliminazione di un nodo figlio selezionato
    def elimina_nodo_figlio(self):
        selected_row = self.tabella_nodi_figlio.currentRow()
        if selected_row >= 0:
            nodo_figlio_id = int(self.tabella_nodi_figlio.item(selected_row, 0).text())
            conn = sqlite3.connect("DATABASE.db")
            if conn:
                cursor = conn.cursor()
                cursor.execute("DELETE FROM nodi_figlio WHERE id=?",
(nodo_figlio_id,))
```

```
            conn.commit()

            conn.close()

            self.carica_nodi_figlio()

    # Gestisce la selezione di un nodo figlio

    def seleziona_nodo_figlio(self, row, col):

        nodo_figlio_id = int(self.tabella_nodi_figlio.item(row, 0).text())

        nome_nodo_figlio = self.tabella_nodi_figlio.item(row, 1).text()

        QMessageBox.information(self, "Nodo Figlio Selezionato", f"ID Nodo Figlio: {nodo_figlio_id}\nNome: {nome_nodo_figlio}")

# Funzione principale

def main():

    app = QApplication(sys.argv)

    window = MappeMentaliApp()

    window.show()

    sys.exit(app.exec_())

if __name__ == "__main__":

    main()
```

Gestore Applicazioni all'Avvio di Windows

Autore: Luca Bocaletto

Il "Gestore Applicazioni all'Avvio di Windows" è un pratico strumento che consente agli utenti di gestire le applicazioni che vengono avviate automaticamente quando il loro computer Windows si avvia. Questa applicazione fornisce un'interfaccia grafica facile da usare per controllare le applicazioni all'avvio.

Caratteristiche Principali

1. **Visualizza e Gestisce le Applicazioni all'Avvio:** Il software mostra un elenco di applicazioni impostate per l'avvio automatico di Windows. Gli utenti possono visualizzare i nomi e i percorsi dei file di queste applicazioni.

2. **Aggiungi Applicazioni all'Avvio:** Gli utenti possono aggiungere nuove applicazioni all'elenco di avvio specificando il percorso del file eseguibile dell'applicazione. Ciò può essere fatto utilizzando il pulsante "Sfoglia" per selezionare l'applicazione.

3. **Rimuovi Applicazioni dall'Avvio:** Le applicazioni indesiderate possono essere rimosse dall'elenco di avvio selezionandole e cliccando sul pulsante "Rimuovi dall'Avvio".

4. **Avvia Applicazioni:** Gli utenti possono avviare qualsiasi applicazione direttamente dall'elenco selezionandola e cliccando sul pulsante "Avvia Applicazione".

5. **Aggiorna l'Elenco:** Il software fornisce un pulsante "Aggiorna l'Elenco" per aggiornare in tempo reale l'elenco delle applicazioni all'avvio.

6. **Interfaccia Utente Intuitiva:** L'applicazione presenta un'interfaccia grafica semplice e intuitiva realizzata utilizzando la libreria tkinter, rendendola accessibile a utenti di tutti i livelli di esperienza.

Uso

- **Aggiungere Applicazioni all'Avvio:** Per aggiungere un'applicazione all'elenco di avvio di Windows, gli utenti possono fare clic sul pulsante "Sfoglia" per selezionare il file eseguibile dell'applicazione. Successivamente, possono fare clic sul pulsante "Aggiungi all'Avvio".

- **Rimuovere Applicazioni dall'Avvio:** Per rimuovere un'applicazione dall'elenco di avvio, gli utenti possono selezionarla nell'elenco e fare clic sul pulsante "Rimuovi dall'Avvio".

- **Avviare Applicazioni:** Gli utenti possono avviare qualsiasi applicazione dall'elenco selezionandola e facendo clic sul pulsante "Avvia Applicazione".

- **Aggiornare l'Elenco:** Il pulsante "Aggiorna l'Elenco" aggiorna l'elenco visualizzato delle applicazioni all'avvio per riflettere eventuali modifiche.

Source Code - Gestore Applicazioni all'Avvio di Windows

```
# Name Software: Windows-Autorun-Process-Manager
# Author: Bocaletto Luca
# Import delle librerie tkinter e altre librerie necessarie

import tkinter as tk
```

```
import tkinter.ttk as ttk  # Importa il modulo ttk da tkinter

import os

import subprocess

import ctypes  # Importa ctypes per la gestione dei messaggi di errore

import winreg  # Importa winreg per la manipolazione del registro di sistema di Windows

from tkinter import filedialog  # Importa il modulo filedialog da tkinter per la finestra di dialogo di selezione file

# Funzione per ottenere le applicazioni all'avvio di Windows

def get_startup_apps():

    startup_apps = []

    # Apre la chiave del registro di sistema HKEY_CURRENT_USER per leggere le applicazioni di avvio

    with winreg.OpenKey(winreg.HKEY_CURRENT_USER, r"Software\Microsoft\Windows\CurrentVersion\Run", 0, winreg.KEY_READ) as key:

        try:

            index = 0

            while True:

                # Enumera i valori nella chiave del registro (nome e percorso dell'applicazione)

                name, value, _ = winreg.EnumValue(key, index)

                startup_apps.append((name, value))

                index += 1

        except WindowsError:
```

```
            pass

    return startup_apps

# Funzione per aggiungere un'applicazione all'avvio di Windows

def aggiungi_startup_app():

    app_path = app_path_entry.get()  # Ottiene il percorso dell'applicazione dall'input dell'utente

    app_name = os.path.basename(app_path)  # Estrae il nome dell'applicazione dal percorso

    # Apre la chiave del registro di sistema per scrivere il nuovo valore (nome e percorso dell'applicazione)

    with winreg.OpenKey(winreg.HKEY_CURRENT_USER, r"Software\Microsoft\Windows\CurrentVersion\Run", 0, winreg.KEY_WRITE) as key:

        winreg.SetValueEx(key, app_name, 0, winreg.REG_SZ, app_path)

    refresh_list()  # Aggiorna la lista delle applicazioni all'avvio

# Funzione per selezionare il percorso dell'applicazione utilizzando il pulsante "Sfoglia"

def seleziona_percorso():

    file_path = filedialog.askopenfilename()  # Apre una finestra di dialogo per la selezione del file

    app_path_entry.delete(0, tk.END)  # Cancella il contenuto dell'entry widget

    app_path_entry.insert(0, file_path)  # Inserisce il percorso del file nell'entry widget


# Funzione per rimuovere un'applicazione dall'avvio di Windows

def rimuovi_startup_app():

    selected_item = startup_apps_list.selection()  # Ottiene l'elemento selezionato nella
```

```
lista

    if selected_item:

        app_name = startup_apps_list.item(selected_item, "values")[0]  # Ottiene il nome dell'applicazione dalla lista

        # Apre la chiave del registro di sistema per rimuovere il valore dell'applicazione dall'avvio

        with winreg.OpenKey(winreg.HKEY_CURRENT_USER, r"Software\Microsoft\Windows\CurrentVersion\Run", 0, winreg.KEY_WRITE) as key:

            winreg.DeleteValue(key, app_name)

        refresh_list()  # Aggiorna la lista delle applicazioni all'avvio

# Funzione per avviare un'applicazione

def avvia_applicazione():

    selected_item = startup_apps_list.selection()  # Ottiene l'elemento selezionato nella lista

    if selected_item:

        app_path = startup_apps_list.item(selected_item, "values")[1]  # Ottiene il percorso dell'applicazione dalla lista

        try:

            subprocess.Popen([app_path], shell=True)  # Avvia l'applicazione tramite subprocess

        except Exception as e:

            # Mostra un messaggio di errore in caso di errore nell'avvio dell'applicazione

            ctypes.windll.user32.MessageBoxW(0, f"Impossibile avviare l'applicazione.\nErrore: {str(e)}", "Errore", 0)

# Funzione per aggiornare la lista delle applicazioni all'avvio
```

```
def refresh_list():
    startup_apps_list.delete(*startup_apps_list.get_children())  # Cancella tutti gli elementi nella lista attuale
    for app_name, app_path in get_startup_apps():
        # Aggiunge le applicazioni all'avvio attuali nella lista
        startup_apps_list.insert("", "end", values=(app_name, app_path))

# Creazione della finestra principale
root = tk.Tk()  # Crea una nuova finestra tkinter
root.title("Gestione Applicazioni all'Avvio di Windows")  # Imposta il titolo della finestra

# Etichetta e campo di inserimento per il percorso dell'applicazione
app_path_label = tk.Label(root, text="Percorso dell'Applicazione:")  # Crea un'etichetta
app_path_label.pack()  # Visualizza l'etichetta nella finestra
app_path_entry = tk.Entry(root)  # Crea un campo di inserimento per il percorso
app_path_entry.pack()  # Visualizza il campo di inserimento nella finestra

# Pulsante "Sfoglia" per selezionare il percorso dell'applicazione
sfoglia_button = tk.Button(root, text="Sfoglia", command=seleziona_percorso)  # Crea un pulsante
sfoglia_button.pack()  # Visualizza il pulsante nella finestra

# Pulsanti per aggiungere, rimuovere e avviare applicazioni
aggiungi_button = tk.Button(root, text="Aggiungi all'Avvio", command=aggiungi_startup_app)  # Crea un pulsante per l'aggiunta
```

rimuovi_button = tk.Button(root, text="Rimuovi dall'Avvio", command=rimuovi_startup_app) # Crea un pulsante per la rimozione

avvia_button = tk.Button(root, text="Avvia Applicazione", command=avvia_applicazione) # Crea un pulsante per l'avvio

aggiungi_button.pack() # Visualizza il pulsante "Aggiungi" nella finestra

rimuovi_button.pack() # Visualizza il pulsante "Rimuovi" nella finestra

avvia_button.pack() # Visualizza il pulsante "Avvia" nella finestra

Lista delle applicazioni all'avvio

startup_apps_list = ttk.Treeview(root, columns=("Nome", "Percorso"), show="headings") # Crea una lista a due colonne

startup_apps_list.heading("Nome", text="Nome") # Imposta l'intestazione della colonna "Nome"

startup_apps_list.heading("Percorso", text="Percorso") # Imposta l'intestazione della colonna "Percorso"

startup_apps_list.pack() # Visualizza la lista nella finestra

refresh_list() # Aggiorna la lista delle applicazioni all'avvio

Esecuzione del loop principale della GUI

root.mainloop() # Avvia il ciclo principale per l'interfaccia grafica

Synthesizer LB-1

Synthesizer LB-1 è un'applicazione software creata da Luca Bocaletto, funge da sintetizzatore MIDI virtuale. Questo software è stato progettato per ricevere input MIDI e generare suoni emulando oscillatori e diverse forme d'onda, consentendo agli utenti di sperimentare e creare musica in modo creativo.

Principali Caratteristiche

- **Interfaccia Utente Intuitiva:** L'applicazione offre un'interfaccia utente intuitiva basata su Tkinter, che permette agli utenti di controllare facilmente il suono generato e i parametri dell'inviluppo.

- **Gestione MIDI:** Synthesizer LB-1 può ricevere input da periferiche MIDI esterne, consentendo agli utenti di suonare il sintetizzatore utilizzando tastiere o controller MIDI.

- **Controllo del Suono:** Gli utenti possono regolare il volume, selezionare diverse forme d'onda (tra cui sinusoide, triangolare, quadra e dente di sega), e modificare i parametri ADSR (Attack, Decay, Sustain, Release) per modellare il suono desiderato.

- **Generazione Audio in Tempo Reale:** Il software genera segnali audio in tempo reale basati sulle interazioni dell'utente, consentendo loro di ascoltare immediatamente il risultato delle modifiche apportate ai parametri.

- **Conversione da Nota MIDI a Frequenza:** Synthesizer LB-1 converte le note MIDI in frequenze corrispondenti, permettendo agli utenti di suonare note specifiche utilizzando input MIDI standard.

- **Compatibilità con Diverse Forme d'Onda:** Oltre alle classiche forme d'onda come il sinusoidale, il software offre forme d'onda più complesse come la sega, la quadra e la triangolare per una vasta gamma di possibilità sonore.

- **Gestione dei Thread:** L'applicazione utilizza thread per gestire la riproduzione audio in modo che l'interfaccia utente rimanga reattiva durante la generazione del suono.

Synthesizer LB-1 è uno strumento ideale per musicisti, compositori e appassionati di sintetizzatori che desiderano sperimentare con la creazione di suoni unici e personalizzati. La sua interfaccia intuitiva e le opzioni di controllo flessibili lo rendono uno strumento potente per l'esplorazione e la produzione musicale.

Source Code - Synthesizer LB-1

```
def on_decay_change(self, decay):
    self.decay_time = float(decay)
def on_sustain_change(self, sustain):
    self.sustain_level = float(sustain)
def on_release_change(self, release):
    self.release_time = float(release)
# Gestore del cambio della forma d'onda
```

```
    def on_waveform_change(self, event):
        selected_waveform = self.waveform_var.get()
        self.current_waveform = selected_waveform
        if self.current_note is not None:
            self.play_note(self.current_note, selected_waveform)
    # Genera un envelope ADSR
    def generate_envelope(self):
        envelope_length = int(44100 * (self.attack_time + self.decay_time +
self.release_time))
        envelope = np.zeros(envelope_length)
        attack_samples = int(44100 * self.attack_time)
        decay_samples = int(44100 * self.decay_time)
        release_samples = int(44100 * self.release_time)

        # Attack
        envelope[:attack_samples] = np.linspace(0, 1, attack_samples)
        # Decay
        envelope[attack_samples:attack_samples + decay_samples] = np.linspace(1,
self.sustain_level, decay_samples
        # Sustain
        envelope[attack_samples + decay_samples:-release_samples] =
self.sustain_level
        # Release
        envelope[-release_samples:] = np.linspace(self.sustain_level, 0,
```

```
release_samples)
        return envelope
    # Genera un segnale audio con envelope
    def generate_signal_with_envelope(self, frequency, envelope, waveform):
        duration = len(envelope) / 44100
        t = np.linspace(0, duration, len(envelope), False)
        signal = self.generate_signal(frequency, self.volume, waveform, duration)
        return signal * envelope
if __name__ == '__main__':
    # Creazione della finestra principale dell'applicazione
    root = tk.Tk()
    app = MidiReceiverApp(root)
    root.mainloop()
```