Obiektowy Python w oknie

Zbigniew Kluczkowski

Spis treści

Czym jest programowanie obiektowe?

Paradygmat programowania oparty na obiektach

Obiekt = dane + funkcje (metody) operujące na tych danych

Główne cechy: hermetyzacja, dziedziczenie, polimorfizm

Klasa i obiekt w Pythonie

```
class Samochod:
    def __init__(self, marka, kolor):
        self.marka = marka
        self.kolor = kolor

    def uruchom(self):
        print(f"{self.marka} rusza!")

auto = Samochod("Toyota", "Czerwony")
auto.uruchom()
```

__init__ – konstruktor klasy

self – odwołanie do konkretnego obiektu

Tworzenie obiektu = wywołanie klasy jak funkcji

Hermetyzacja (Enkapsulacja)

Ukrywanie wewnętrznych szczegółów obiektu.

```
class KontoBankowe:
    def __init__(self):
        self.__saldo = 0

    def wplata(self, kwota):
        self.__saldo += kwota

    def pokaz_saldo(self):
        return self.__saldo
```

__saldo – pole prywatne (niedostępne bezpośrednio)

Dziedziczenie

Pozwala tworzyć nowe klasy na podstawie istniejących.

```
class Zwierze:
    def dzwiek(self):
        print("Dźwięk zwierzęcia")

class Pies(Zwierze):
    def dzwiek(self):
        print("Hau hau!")

p = Pies()
p.dzwiek()
```

Klasa Pies dziedziczy z klasy Zwierze, ale może nadpisywać metody

Polimorfizm

Ta sama metoda, różne zachowanie w zależności od obiektu.

```
def wydaj_dzwiek(zwierze):
   zwierze.dzwiek()

wydaj_dzwiek(Pies())
wydaj_dzwiek(Kot())
```

Funkcja wydaj_dzwiek działa na różnych typach obiektów.

Klasy i metody statyczne

Metody statyczne nie wymagają obiektu.

```
class Matematyka:
    @staticmethod
    def dodaj(a, b):
        return a + b

print(Matematyka.dodaj(3, 5))
```

Dlaczego warto używać OOP w Pythonie?

Lepsza organizacja kodu

Ułatwia rozwój i utrzymanie

Współpraca wielu programistów nad jednym projektem

Wspiera dobre praktyki projektowe

Podstawowe biblioteki w Pythonie

Biblioteka	Opis
math	Funkcje matematyczne, np. pierwiastki, funkcje trygonometryczne.
random	Generowanie liczb losowych.
datetime	Operacje na datach i czasie.
os	Operacje na plikach i katalogach systemowych.
sys	Dostęp do argumentów wiersza poleceń i środowiska uruchomieniowego.
json	Odczyt/zapis danych w formacie JSON.
re	Operacje na wyrażeniach regularnych.
tkinter	Tworzenie GUI (graficzny interfejs użytkownika).

Biblioteki matematyczne w Pythonie

Biblioteka	Opis
numpy	Operacje na tablicach, funkcje matematyczne (wydajna matematyka).
scipy	Zaawansowane funkcje naukowe i inżynieryjne (np. całkowanie, optymalizacja).
pandas	Analiza i manipulacja danymi w tabelach (DataFrame).

Biblioteki do prezentacji danych w Pythonie

Biblioteka	Opis
matplotlib	Tworzenie wykresów.
seaborn	Zaawansowane wykresy, oparte na matplotlib.
plotly	Interaktywne wykresy.

Uczenie maszynowe i AI w Pythonie

Biblioteka	Opis
scikit-learn	Klasyczne algorytmy uczenia maszynowego.
tensorflow	Głębokie uczenie (deep learning).
keras	Prostszym interfejs do TensorFlow.
pytorch	Alternatywa dla TensorFlow, używana w badaniach naukowych.

Praca z internetem i API w Pythonie

BIBLIOTEKA	OPIS
requests	Pobieranie danych z Internetu (HTTP).
beautifulsoup4	Parsowanie HTML, web scraping.
urllib	Biblioteka standardowa do pracy z adresami URL.

Praca z bazami danych w Pythonie

Biblioteka	Opis
csv	Odczyt/zapis plików CSV.
openpyxl / xlrd	Praca z plikami Excel.
sqlite3	Baza danych SQLite wbudowana w Pythona.

Popularne biblioteki Pythona wykorzystujące OOP

Python to nie tylko język – to całe ekosystemy bibliotek zbudowanych w stylu obiektowym:

- **Django** – framework webowy
- **Flask** – lekki framework webowy
- **Pygame** – tworzenie gier
- **Tkinter** – GUI w Pythonie
- **Pandas** – analiza danych
- **SQLAlchemy** – obiektowa obsługa baz danych

Django – Web development w OOP

Wszystko w Django to **klasy**: modele, widoki, formularze

Przykład modelu:

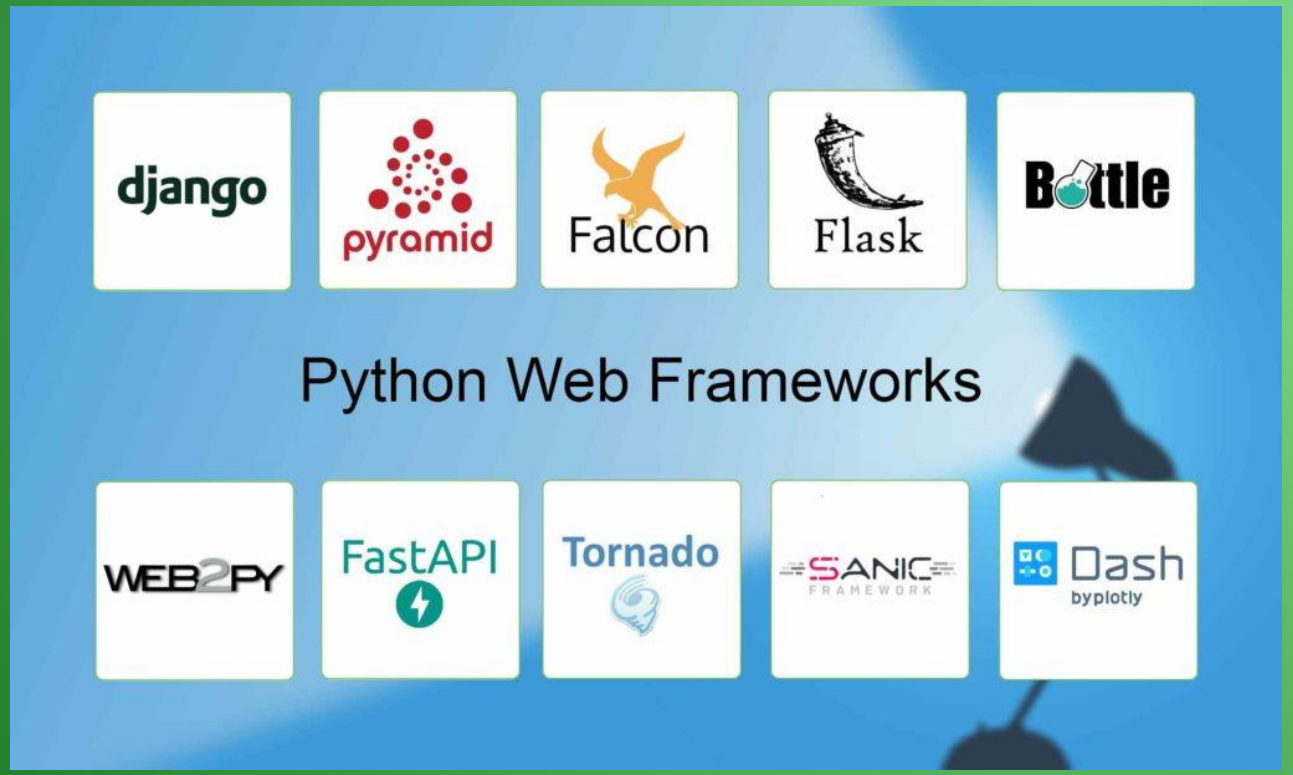
```
from django.db import models

class Post(models.Model):
    tytul = models.CharField(max_length=100)
    tresc = models.TextField()
    data = models.DateTimeField(auto_now_add=True)
```

Każda klasa modelu odwzorowuje tabelę w bazie danych.

Flask – prosty web framework

Choć minimalistyczny,
Flask też wspiera OOP.

Można tworzyć aplikacje jako klasy:




```
from flask import Flask

class MyApp:
    def __init__(self):
        self.app = Flask(__name__)
        self.setup_routes()

    def setup_routes(self):
        @self.app.route('/')
        def home():
            return "Witaj!"

app = MyApp().app
app.run()
```

Pygame – gry obiektowo

```
import pygame


class Gracz(pygame.sprite.Sprite):
    def __init__(self):
        super().__init__()
        self.image = pygame.Surface((50,
50))
        self.image.fill((255, 0, 0))
        self.rect = self.image.get_rect()


    def ruch(self):
        self.rect.x += 5
```

Postacie, obiekty, tło – wszystko to klasy i obiekty.

Tkinter – graficzne interfejsy

```
import tkinter as tk


class Aplikacja(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("Moja aplikacja")
        self.label = tk.Label(self,
text="Witaj!")
        self.label.pack()


app = Aplikacja()
app.mainloop()
```

Dlaczego Tkinter?

Łatwość w użyciu: Tkinter jest jednym z najprostszych narzędzi do tworzenia GUI w Pythonie.

Dostępność: Tkinter jest częścią standardowej biblioteki Pythona, więc jest dostępny bezpośrednio po zainstalowaniu Pythona.

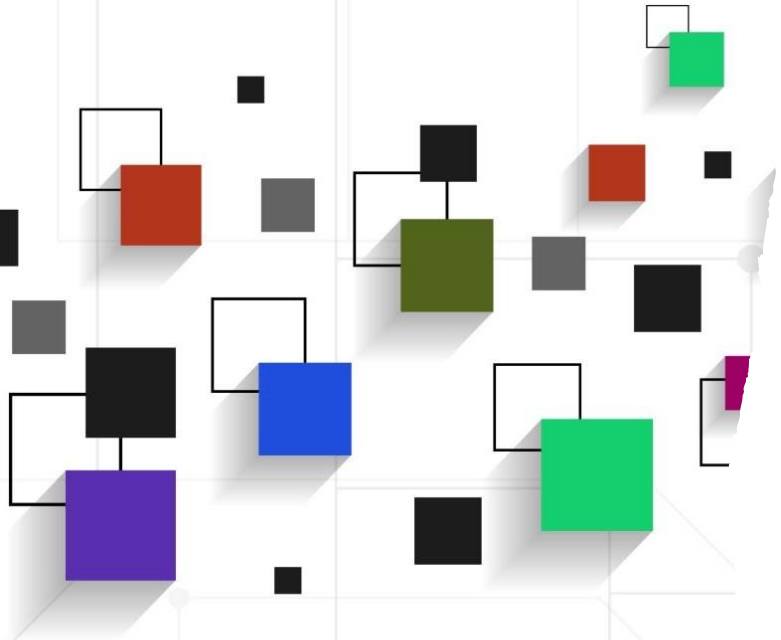
Wszechstronność: Tkinter pozwala na tworzenie różnych typów aplikacji - od prostych narzędzi po bardziej złożone interfejsy.

Dokumentacja i społeczność: Jako część Pythona, Tkinter ma dużą społeczność i bogatą dokumentację.

Przykładowy kod Tkinter

```
import tkinter as tk


root = tk.Tk()
root.title("Moja aplikacja")


root.mainloop()
```

- **tk.Tk()**: Tworzy główne okno aplikacji.
- **root.mainloop()**: Uruchamia pętlę, która sprawia, że aplikacja pozostaje aktywna i reaguje na zdarzenia użytkownika.



Dodawanie widgetów

Tkinter oferuje różne **widgety**, które można dodać do okna:

- **Label**: Etykieta (napis)
- **Button**: Przycisk
- **Entry**: Pole tekstowe
- **Text**: Pole wieloliniowe
- **Canvas**: Obszar do rysowania
- **Frame**: Ramka do organizowania widgetów

Podstawowe widgety interfejsu

Widget	Opis
Button	Przycisk
Label	Etykieta tekstowa (statyczna)
Entry	Jednoliniowe pole do wpisywania tekstu
Text	Wieloliniowe pole tekstowe
Checkbutton	Pole wyboru (checkbox)
Radiobutton	Przycisk opcji (radio)
Spinbox	Pole z wartościami numerycznymi do przewijania
Scale	Suwak do wyboru wartości liczbowej
Listbox	Lista elementów do wyboru
Combobox	Rozwijane menu (z ttk)
Scrollbar	Pasek przewijania
Menu	Pasek menu górnego (np. Plik, Edycja)
Message	Jak Label, ale obsługuje dłuższy tekst z łamaniem wierszy

Kontenery i układ

Widget	Opis
Frame	Kontener dla grupowania innych widgetów
LabelFrame	Frame z etykietą tytułową
PanedWindow	Przestrzeń podzielona na przesuwalne panele
Toplevel	Nowe okno (poza głównym)
Canvas	Obszar rysowania (linie, figury, obrazy)
Notebook	Zakładki (z ttk)
Treeview	Widok drzewa/tabeli (z ttk)
Separator	Linia oddzielająca (z ttk)
Progressbar	Pasek postępu (z ttk)

Multimedia i rysowanie

Widget	Opis
Canvas	Rysowanie, obrazy, figury, animacje
PhotoImage	Wczytywanie i wyświetlanie obrazów (np. w Label lub Button)
Image (PIL)	Obsługa bardziej zaawansowana (JPEG, PNG) – wymaga Pillow

Przykładowy przycisk z obsługą

```
def przycisk_clicked():
    print("Przycisk kliknięty!")


przycisk = tk.Button(root,
text="Kliknij mnie",
command=przycisk_clicked)
przycisk.pack()
```

- Prosta funkcja (def) wyświetlająca napis po naciśnięciu przycisku. Command wywołuje funkcję.
- Pack ustawia elementy jeden pod drugim.

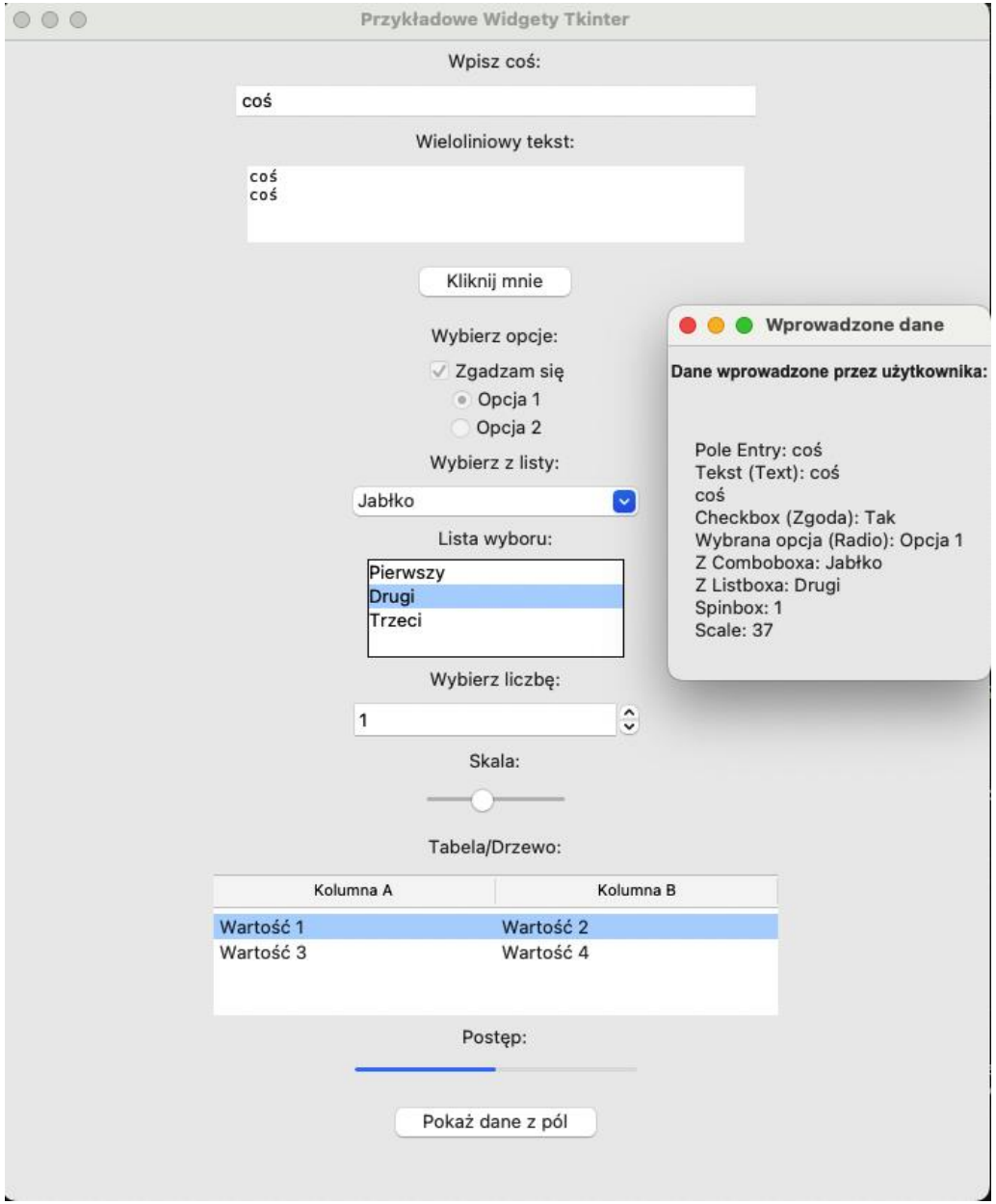
Przykładowy program w Tkinter

```
Import tkinter as tk
from tkinter import ttk
from tkinter import messagebox

def show_info():
messagebox.showinfo("Informacja", "Kliknięto przycisk!")

def show_user_input():
# Pobieranie danych z pól
entry_val = entry.get()
text_val = text.get("1.0", "end").strip()
check_val = check_var.get()
radio_val = radio_var.get()
combo_val = combo.get()
listbox_selection = listbox.get(tk.ACTIVE)
spin_val = spin.get()
scale_val = int(scale.get())

# Tworzenie nowego okna z podsumowaniem
result_win = tk.Toplevel(root)
result_win.title("Wprowadzone dane")

output = f"""
Pole Entry: {entry_val}
Tekst (Text): {text_val}
Checkbox (Zgoda): {"Tak" if check_val else "Nie"}
Wybrana opcja (Radio): {radio_val}
Z Comboboxa: {combo_val}
Z Listboxa: {listbox_selection}
Spinbox: {spin_val}
Scale: {scale_val}
"""

# Wyświetlanie w Labelu w nowym oknie
ttk.Label(result_win, text="Dane wprowadzone przez użytkownika:", font=("Arial", 12, "bold")).pack(pady=10)
ttk.Label(result_win, text=output, justify="left").pack(padx=10, pady=10)

# Główne okno
root = tk.Tk()
root.title("Przykładowe Widgety Tkinter")
root.geometry("700x650")

# === Sekcja: Label i Entry ===
ttk.Label(root, text="Wpisz coś:").pack(pady=5)
entry = ttk.Entry(root, width=40)
entry.pack()

# === Sekcja: Text ===
ttk.Label(root, text="Wieloliniowy tekst:").pack(pady=5)
text = tk.Text(root, height=4, width=50)
text.pack()
```

Przykładowy program w Tkinter

```
# === Sekcja: Button ===
ttk.Button(root, text="Kliknij mnie", command=show_info).pack(pady=10)

# === Sekcja: Checkbutton i Radiobutton ===
ttk.Label(root, text="Wybierz opcje:").pack(pady=5)

check_var = tk.BooleanVar()
ttk.Checkbutton(root, text="Zgadzam się", variable=check_var).pack()

radio_var = tk.StringVar(value="Opcja 1")
ttk.Radiobutton(root, text="Opcja 1", variable=radio_var, value="Opcja 1").pack()
ttk.Radiobutton(root, text="Opcja 2", variable=radio_var, value="Opcja 2").pack()

# === Sekcja: Combobox ===
ttk.Label(root, text="Wybierz z listy:").pack(pady=5)
combo = ttk.Combobox(root, values=["Jabłko", "Banan", "Pomarańcza"])
combo.pack()

# === Sekcja: Listbox ===
ttk.Label(root, text="Lista wyboru:").pack(pady=5)
listbox = tk.Listbox(root, height=4)
for item in ["Pierwszy", "Drugi", "Trzeci"]:
listbox.insert(tk.END, item)
listbox.pack()

# === Sekcja: Spinbox ===
ttk.Label(root, text="Wybierz liczbę:").pack(pady=5)
spin = tk.Spinbox(root, from_=0, to=10)
spin.pack()

# === Sekcja: Scale ===
ttk.Label(root, text="Skala:").pack(pady=5)
scale = ttk.Scale(root, from_=0, to=100, orient="horizontal")
scale.pack()

# === Sekcja: Treeview ===
ttk.Label(root, text="Tabela/Drzewo:").pack(pady=10)
tree = ttk.Treeview(root, columns=("A", "B"), show="headings", height=4)
tree.heading("A", text="Kolumna A")
tree.heading("B", text="Kolumna B")
tree.insert("", "end", values=("Wartość 1", "Wartość 2"))
tree.insert("", "end", values=("Wartość 3", "Wartość 4"))
tree.pack()

# === Sekcja: Pasek postępu ===
ttk.Label(root, text="Postęp:").pack(pady=5)
progress = ttk.Progressbar(root, value=50, maximum=100, length=200)
progress.pack()

# === Przyciski akcji ===
ttk.Button(root, text="Pokaż dane z pól", command=show_user_input).pack(pady=15)

root.mainloop()
```

Wygląd i funkcjonalność kodu



Opis działania programu:

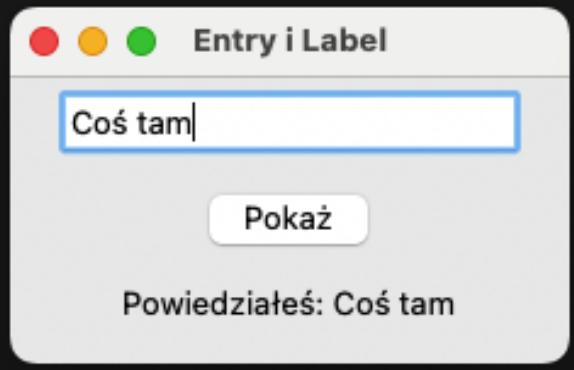
Użytkownik prowadza dane do pól tekstowych, wybiera opcje oraz może wybrać dane z tabeli. Następnie program w osobnym oknie wyświetla dane wprowadzone za pomocą formularza.

Proste? Na tym przykładzie można ćwiczyć inne formularze, których zadaniem jest proste gromadzenie i wyświetlanie danych.

Gromadzenie danych w listach i tabelach z wykorzystaniem kolekcji danych, plików oraz bazy danych zobaczysz w dalszej części prezentacji.

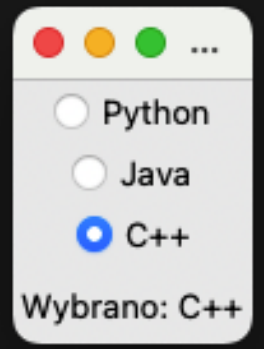
Widżety wraz z funkcjami – Entry, Button i Label

```python
import tkinter as tk

def pokaz_tekst():
    tekst = entry.get()
    label_wynik.config(text=f"Powiedziałeś: {tekst}")

root = tk.Tk()
root.title("Entry i Label")

entry = tk.Entry(root)
entry.pack(pady=5)

btn = tk.Button(root, text="Pokaż", command=pokaz_tekst)
btn.pack(pady=5)

label_wynik = tk.Label(root, text="")
label_wynik.pack(pady=5)

root.mainloop()
```



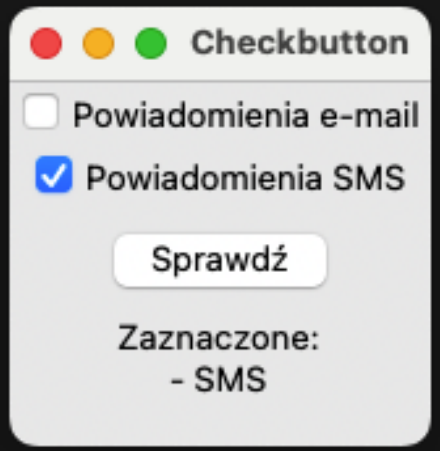
Widżety wraz z funkcjami – RadioButton, Label i Button

```python
import tkinter as tk

def pokaz_wybor():
    label_wynik.config(text=f"Wybrano: {opcja.get()}")

root = tk.Tk()
root.title("Radiobutton")

opcja = tk.StringVar(value="Brak")

tk.Radiobutton(root, text="Python", variable=opcja, value="Python", command=pokaz_wybor).pack()
tk.Radiobutton(root, text="Java", variable=opcja, value="Java", command=pokaz_wybor).pack()
tk.Radiobutton(root, text="C++", variable=opcja, value="C++", command=pokaz_wybor).pack()

label_wynik = tk.Label(root, text="Wybrano: Brak")
label_wynik.pack(pady=5)

root.mainloop()
```



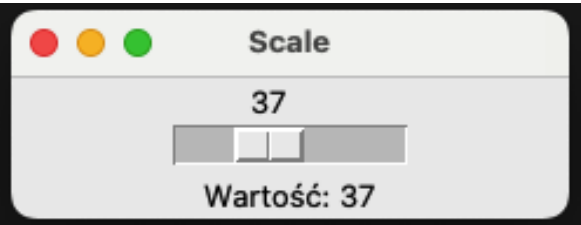
Widżety wraz z funkcjami – Checkbox, Label i Button

```python
import tkinter as tk

def sprawdz():
    wynik = "Zaznaczone:\n"
    if var1.get(): wynik += "- E-mail\n"
    if var2.get(): wynik += "- SMS\n"
    label_wynik.config(text=wynik)

root = tk.Tk()
root.title("Checkbutton")

var1 = tk.BooleanVar()
var2 = tk.BooleanVar()

tk.Checkbutton(root, text="Powiadomienia e-mail", variable=var1).pack()
tk.Checkbutton(root, text="Powiadomienia SMS", variable=var2).pack()

tk.Button(root, text="Sprawdź", command=sprawdz).pack(pady=5)
label_wynik = tk.Label(root, text="")
label_wynik.pack()

root.mainloop()
```



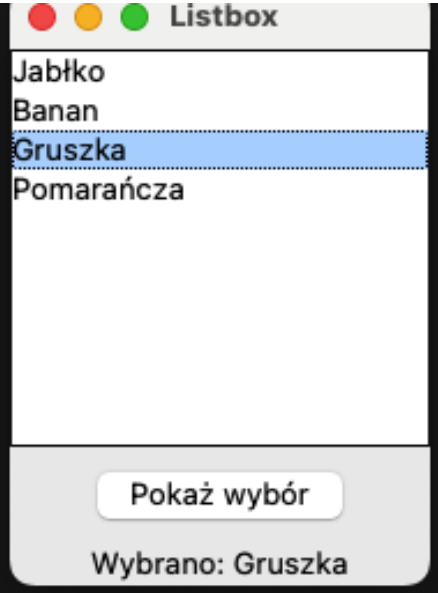
Widżety wraz z funkcjami – Scale I Label

```python
import tkinter as tk

def aktualizuj_wartosc(val):
    label.config(text=f"Wartość: {val}")

root = tk.Tk()
root.title("Scale")

suwak = tk.Scale(root, from_=0, to=100, orient="horizontal", command=aktualizuj_wartosc)
suwak.pack()

label = tk.Label(root, text="Wartość: 0")
label.pack()

root.mainloop()
```



Widżety wraz z funkcjami – Listbox, Button i Label

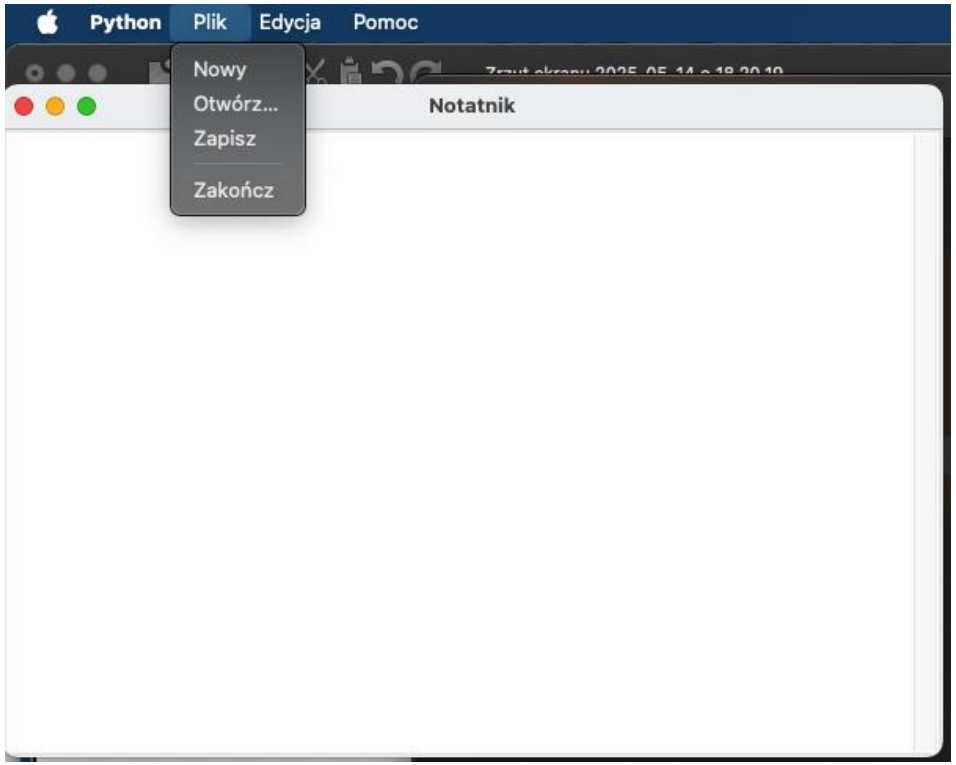
```python
import tkinter as tk

def pokaz_wybor():
    indeks = lista.curselection()
    if indeks:
        label.config(text=f"Wybrano: {lista.get(indeks)}")

root = tk.Tk()
root.title("Listbox")

lista = tk.Listbox(root)
for owoc in ["Jabłko", "Banan", "Gruszka", "Pomarańcza"]:
    lista.insert(tk.END, owoc)
lista.pack()

tk.Button(root, text="Pokaż wybór", command=pokaz_wybor).pack(pady=5)
label = tk.Label(root, text="")
label.pack()

root.mainloop()
```

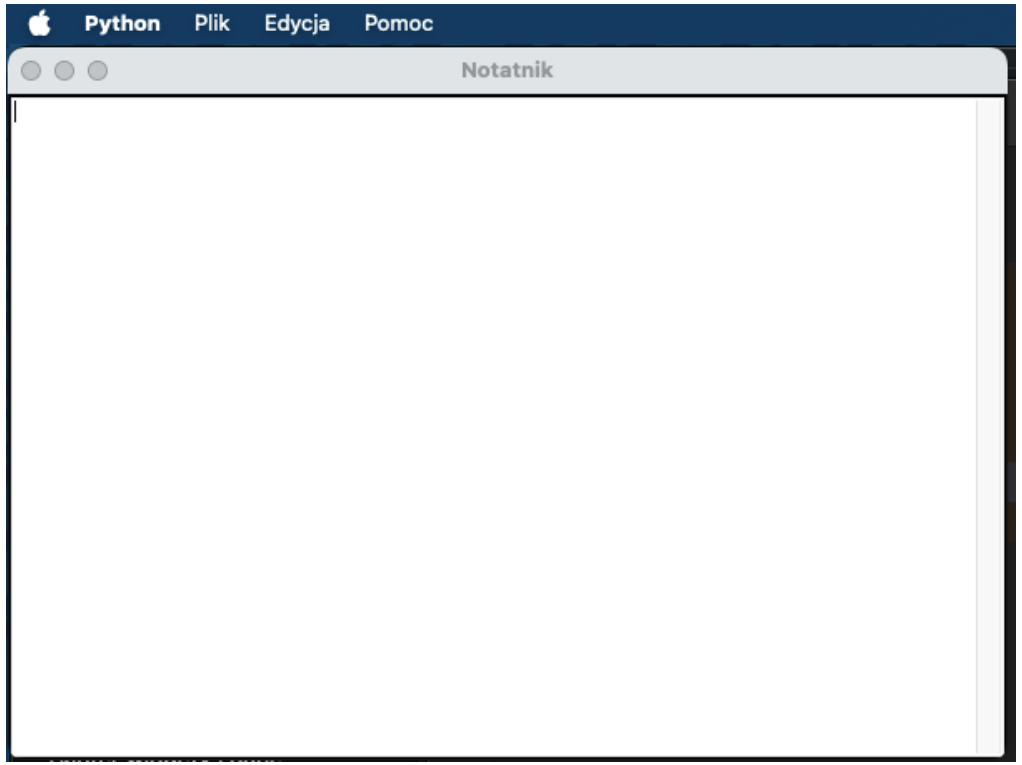


Widżety wraz z funkcjami - Menu

```
import tkinter as tk

from tkinter import filedialog, messagebox


def new_file():

    text.delete("1.0", tk.END)

    root.title("Nowy - Notatnik")


def open_file():

    file = filedialog.askopenfilename(filetypes=[("Pliki tekstowe", "*.txt")])

    if file:

        with open(file, "r") as f:

            text.delete("1.0", tk.END)

            text.insert(tk.END, f.read())

        root.title(f"{file} - Notatnik")


def save_file():

    file = filedialog.asksaveasfilename(defaultextension=".txt",

                                        filetypes=[("Pliki tekstowe", "*.txt")])

    if file:

        with open(file, "w") as f:

            f.write(text.get("1.0", tk.END))

        root.title(f"{file} - Notatnik")


def show_about():

    messagebox.showinfo("O programie", "Prosty notatnik w Tkinter\nAutor: Ty 😁")


root = tk.Tk()

root.title("Notatnik")

root.geometry("600x400")


# Menu górne

menu_bar = tk.Menu(root)

root.config(menu=menu_bar)


# --- Menu Plik ---

file_menu = tk.Menu(menu_bar, tearoff=0)

menu_bar.add_cascade(label="Plik", menu=file_menu)

file_menu.add_command(label="Nowy", command=new_file)

file_menu.add_command(label="Otwórz...", command=open_file)

file_menu.add_command(label="Zapisz", command=save_file)

file_menu.add_separator()

file_menu.add_command(label="Zakończ", command=root.quit)


# --- Menu Edycja ---

edit_menu = tk.Menu(menu_bar, tearoff=0)

menu_bar.add_cascade(label="Edycja", menu=edit_menu)

edit_menu.add_command(label="Cofnij", command=lambda: text.edit_undo())

edit_menu.add_separator()

edit_menu.add_command(label="Wytnij", command=lambda: text.event_generate("<<Cut>>"))

edit_menu.add_command(label="Kopiuj", command=lambda: text.event_generate("<<Copy>>"))

edit_menu.add_command(label="Wklej", command=lambda: text.event_generate("<<Paste>>"))

edit_menu.add_separator()

edit_menu.add_command(label="Zaznacz wszystko", command=lambda: text.tag_add(tk.SEL, "1.0", tk.END))


# --- Menu Pomoc ---

help_menu = tk.Menu(menu_bar, tearoff=0)

menu_bar.add_cascade(label="Pomoc", menu=help_menu)

help_menu.add_command(label="O programie", command=show_about)


# Pole tekstowe z przewijaniem

text = tk.Text(root, undo=True, wrap="word")

text.pack(expand=True, fill=tk.BOTH)


scrollbar = tk.Scrollbar(text)

scrollbar.pack(side=tk.RIGHT, fill=tk.Y)

scrollbar.config(command=text.yview)

text.config(yscrollcommand=scrollbar.set)


root.mainloop()
```

Notatnik wykorzystujący opcje w Menu





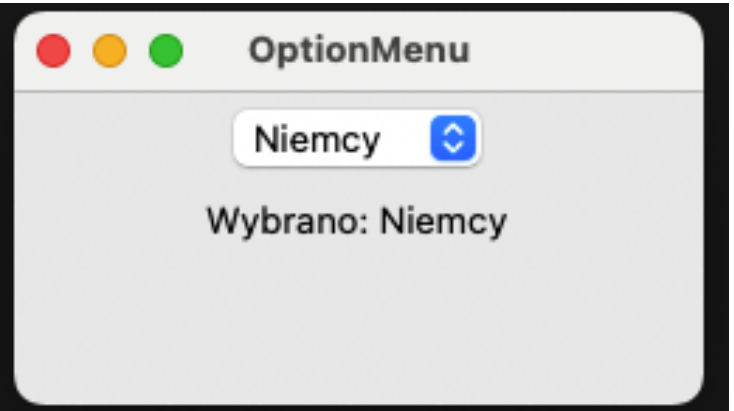
Opis działania programu

Jak w większości języków – podstawowe funkcje są już zdefiniowane (wytnij, kopiuj, wklej, zapisz etc.). Wystarczy je tylko podłączyć pod właściwy element.

Co jest „nie tak” na tych zdjęciach? W MacOS pasek menu przytwierdzi się do górnej ”belki”. W Windows jest zawsze elementem okna.

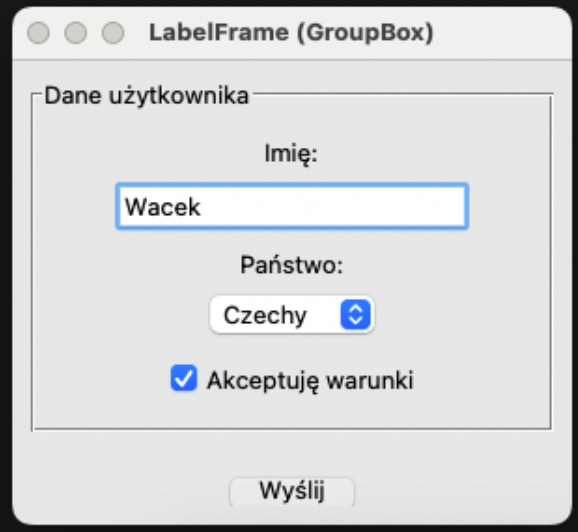
Widżety wraz z funkcjami – OptionMenu i Label

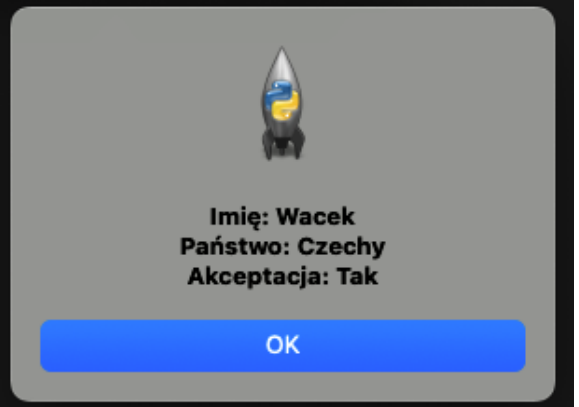
```python
import tkinter as tk

def pokaz_wybor(*args):
    label.config(text=f"Wybrano: {wybor.get()}")

root = tk.Tk()
root.title("OptionMenu")

wybor = tk.StringVar(value="Wybierz...")
wybor.trace("w", pokaz_wybor)

op_menu = tk.OptionMenu(root, wybor, "Polska", "Niemcy", "Francja", "Włochy")
op_menu.pack(pady=5)

label = tk.Label(root, text="Wybrano: ---")
label.pack()

root.mainloop()
```



Widżety wraz z funkcjami – GroupBox (LabelFrame)

```python
import tkinter as tk

def pokaz_dane():
    wynik = f"Imię: {entry.get()}\n"
    wynik += f"Państwo: {menu_var.get()}\n"
    wynik += f"Akceptacja: {'Tak' if check_var.get() else 'Nie'}"
    tk.messagebox.showinfo("Dane", wynik)

root = tk.Tk()
root.title("LabelFrame (GroupBox)")
root.geometry("300x250")

# Import dla messagebox
from tkinter import messagebox

frame = tk.LabelFrame(root, text="Dane użytkownika", padx=10, pady=10)
frame.pack(padx=10, pady=10, fill="both", expand=True)

tk.Label(frame, text="Imię:").pack()
entry = tk.Entry(frame)
entry.pack(pady=5)

tk.Label(frame, text="Państwo:").pack()
menu_var = tk.StringVar(value="Polska")
tk.OptionMenu(frame, menu_var, "Polska", "Czechy", "Słowacja", "Węgry").pack(pady=5)

check_var = tk.BooleanVar()
tk.Checkbutton(frame, text="Akceptuję warunki", variable=check_var).pack(pady=5)

tk.Button(root, text="Wyślij", command=pokaz_dane).pack(pady=10)

root.mainloop()
```

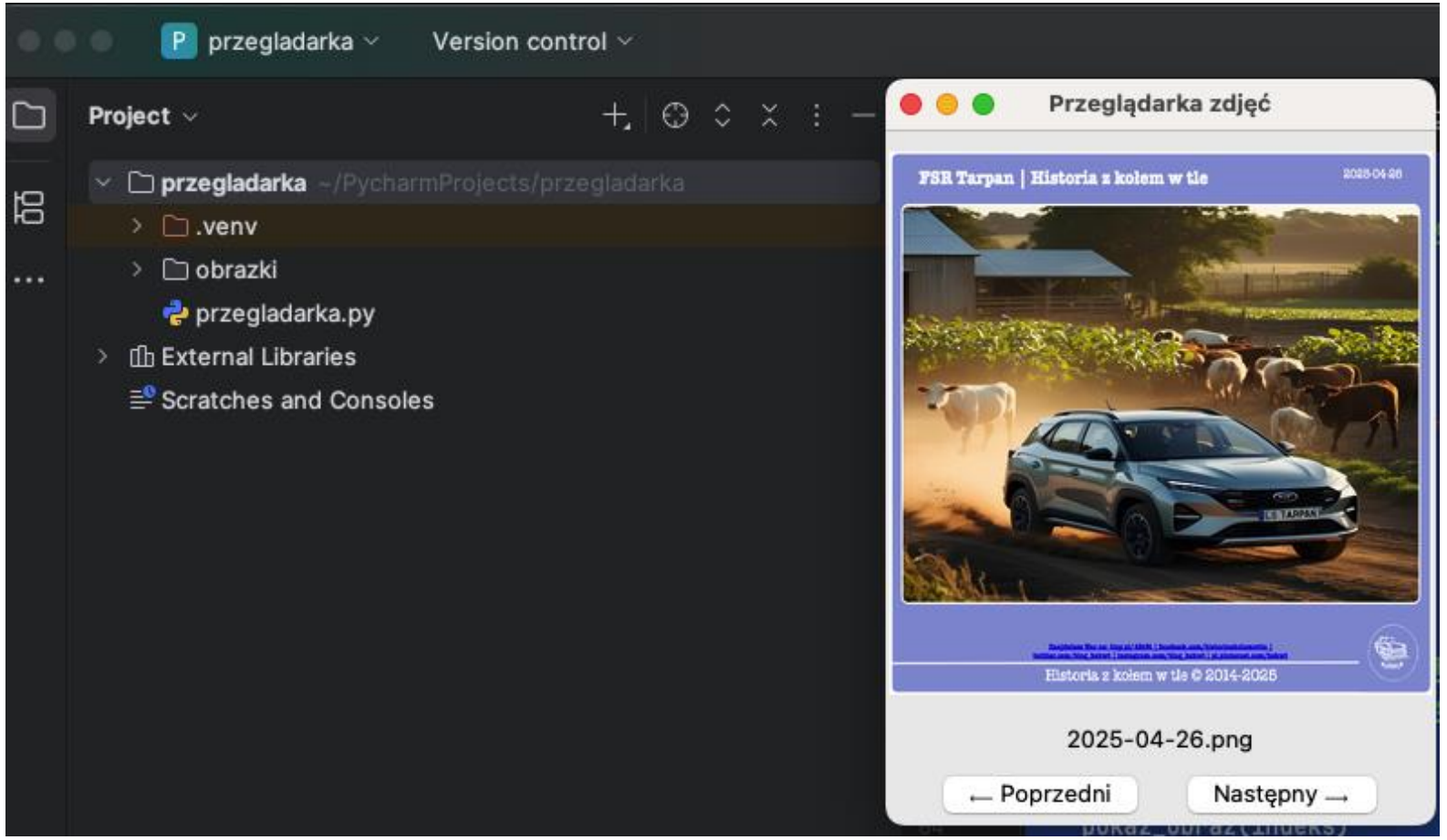




Widżety wraz z funkcjami – przeglądarka obrazków (kod)

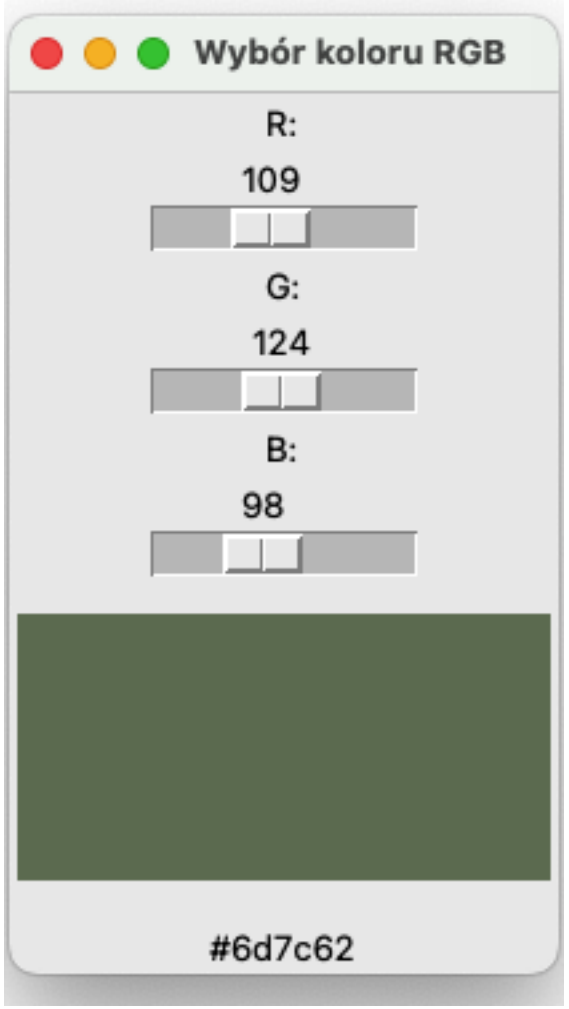
```
import tkinter as tk
from PIL import Image, ImageTk
import os

# Ścieżka do folderu z obrazkami
FOLDER_OBRAZOW = os.path.join(os.path.dirname(__file__), "obrazki")

# Lista dostępnych plików graficznych
lista_plikow = [f for f in os.listdir(FOLDER_OBRAZOW)
            if f.lower().endswith((".jpg", ".jpeg", ".png", ".gif"))]

print("Znalezione obrazy:", lista_plikow)

indeks = 0

# Funkcja do ładowania i wyświetlania obrazu
def pokaz_obraz(ind):
  global img  # <-- TO JEST KLUCZOWE! bez tego obraz może zniknąć

  try:
    sciezka = os.path.join(FOLDER_OBRAZOW, lista_plikow[ind])
    print("Ładowanie:", sciezka)

    obraz = Image.open(sciezka)
    obraz = obraz.resize((300, 300), Image.Resampling.LANCZOS)  #
zmniejszenie obrazu
    img = ImageTk.PhotoImage(obraz)

    label.config(image=img)
    label.image = img  # <- ZACHOWAJ REFERENCJĘ
    label_nazwa.config(text=lista_plikow[ind])

  except Exception as e:
    label.config(text="Błąd ładowania obrazu")
    print("Błąd:", e)

# Obsługa przycisków
def nastepny():
  global indeks
  indeks = (indeks + 1) % len(lista_plikow)
  pokaz_obraz(indeks)

def poprzedni():
  global indeks
  indeks = (indeks - 1) % len(lista_plikow)
  pokaz_obraz(indeks)

# Tworzenie GUI
root = tk.Tk()
root.title("Przeglądarka zdjęć")

label = tk.Label(root)
label.pack(pady=10)

label_nazwa = tk.Label(root, text="")
label_nazwa.pack()

btn_frame = tk.Frame(root)
btn_frame.pack(pady=5)

tk.Button(btn_frame, text="⟵ Poprzedni",
command=poprzedni).pack(side="left", padx=10)
tk.Button(btn_frame, text="Następny ⟶",
command=nastepny).pack(side="left", padx=10)

if lista_plikow:
  pokaz_obraz(indeks)
else:
  label.config(text="Brak obrazów w folderze 'obrazki/'")
  print("‼️ Brak obrazów!")

root.mainloop()
```

Przeglądarka obrazków – efekt działania kodu



Widżety wraz z funkcjami – Tk.scale

```
import tkinter as tk

def aktualizuj():
    r = r_var.get()
    g = g_var.get()
    b = b_var.get()
    kolor = f"#{r:02x}{g:02x}{b:02x}"
    canvas.config(bg=kolor)
    label_kolor.config(text=kolor)

root = tk.Tk()
root.title("Wybór koloru RGB")

r_var = tk.IntVar()
g_var = tk.IntVar()
b_var = tk.IntVar()

tk.Label(root, text="R:").pack()
tk.Scale(root, from_=0, to=255, orient="horizontal", variable=r_var,
command=lambda x: aktualizuj()).pack()

tk.Label(root, text="G:").pack()
tk.Scale(root, from_=0, to=255, orient="horizontal", variable=g_var,
command=lambda x: aktualizuj()).pack()

tk.Label(root, text="B:").pack()
tk.Scale(root, from_=0, to=255, orient="horizontal", variable=b_var,
command=lambda x: aktualizuj()).pack()

canvas = tk.Canvas(root, width=200, height=100)
canvas.pack(pady=10)

label_kolor = tk.Label(root, text="#000000")
label_kolor.pack()

root.mainloop()
```



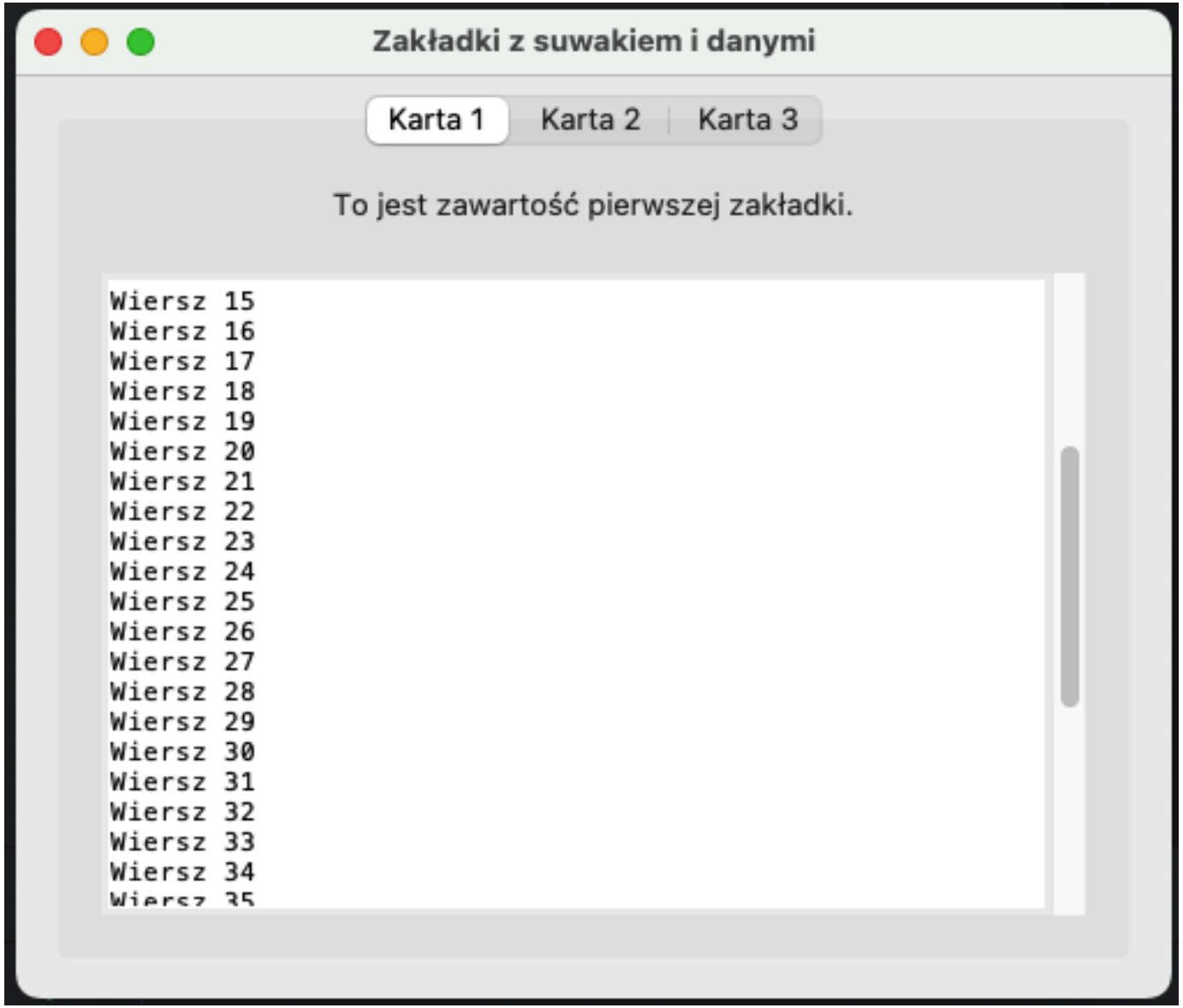
Program przedstawia w oknie kolory powstałe za pomocą odczytu danych z suwaków.

Okno, karty, suwak i zbieranie informacji

```
import tkinter as tk
from tkinter import ttk

# Główne okno
root = tk.Tk()
root.title("Zakładki z suwakiem i danymi")
root.geometry("500x400")

# Tworzenie zakładek
notebook = ttk.Notebook(root)
notebook.pack(fill="both", expand=True)

# --- Karta 1 ---
frame1 = ttk.Frame(notebook)
notebook.add(frame1, text="Karta 1")

label1 = ttk.Label(frame1, text="To jest zawartość pierwszej zakładki.")
label1.pack(pady=10)

# Suwak i pole tekstowe
text_frame = ttk.Frame(frame1)
text_frame.pack(fill="both", expand=True, padx=10, pady=10)

scrollbar = tk.Scrollbar(text_frame)
scrollbar.pack(side="right", fill="y")

text = tk.Text(text_frame, yscrollcommand=scrollbar.set, wrap="word")
text.pack(side="left", fill="both", expand=True)

scrollbar.config(command=text.yview)

for i in range(50):
    text.insert("end", f"Wiersz {i+1}\n")

# --- Karta 2 ---
frame2 = ttk.Frame(notebook)
notebook.add(frame2, text="Karta 2")

label2 = ttk.Label(frame2, text="Wpisz dane:")
label2.pack(pady=10)

entry = ttk.Entry(frame2)
entry.pack(pady=5)

# --- Karta 3 ---
frame3 = ttk.Frame(notebook)
notebook.add(frame3, text="Karta 3")

wynik_label = ttk.Label(frame3, text="Tu pojawią się dane z Karty 2")
wynik_label.pack(pady=20)

def pokaz_dane():
    dane = entry.get()
    if dane.strip():
        wynik_label.config(text=f"Wpisano: {dane}")
    else:
        wynik_label.config(text="Nie wpisano żadnych danych.")

btn_pokaz = ttk.Button(frame3, text="Pokaż dane z Karty 2",
command=pokaz_dane)
btn_pokaz.pack(pady=10)

# Uruchomienie aplikacji
root.mainloop()
```

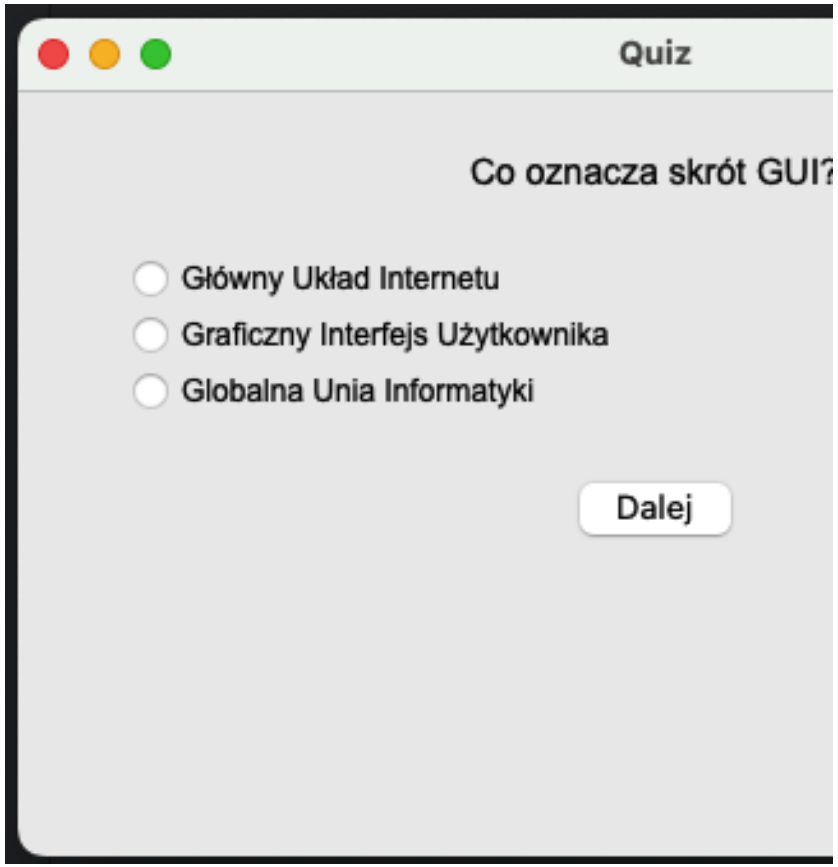
Widok i działanie programu

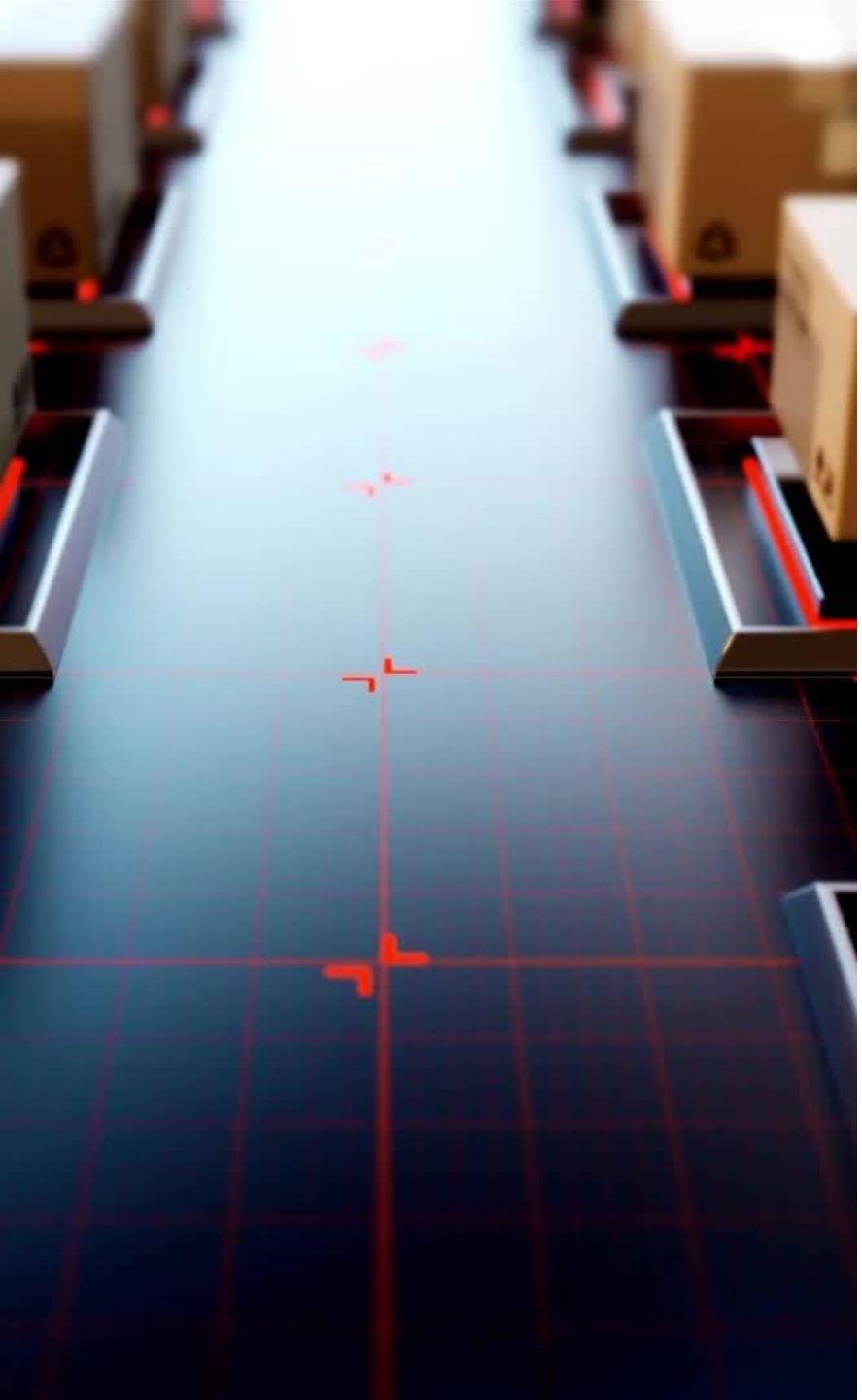
Pierwsza karta zawiera listę rozwijaną z suwakiem. Druga zawiera pole do wpisania danych. Po wpisaniu danych użytkownik przechodzi na trzecią kartę i po naciśnięciu przycisku może wyświetlić dane z drugiej karty.



Quiz – program w formie sprawdzianu

```
import tkinter as tk
from tkinter import messagebox

# Dane
pytania = [
  {
    "pytanie": "Co oznacza skrót GUI?",
    "odpowiedzi": ["Główny Układ Internetu", "Graficzny Interfejs Użytkownika", "Globalna Unia Informatyki"],
    "poprawna": 1
  }
]

indeks_pytania = 0
punkty = 0

# Funkcje
def wyswietl_pytanie():
  pytanie = pytania[indeks_pytania]

label_pytanie.config(text=pytanie["pytanie"])
  for i in range(len(pytanie["odpowiedzi"])):

radio[i].config(text=pytanie["odpowiedzi"][i])
  var.set(-1)

def dalej():
  global indeks_pytania, punkty
  if var.get() == -1:

messagebox.showwarning("Uwaga", "Wybierz odpowiedź!")
    return

  if var.get() == pytania[indeks_pytania]["poprawna"]:
    punkty += 1

  indeks_pytania += 1
  if indeks_pytania < len(pytania):
    wyswietl_pytanie()
  else:

messagebox.showinfo("Koniec", f"Wynik: {punkty}/{len(pytania)}")
    root.quit()

# GUI
root = tk.Tk()
root.title("Quiz")
root.geometry("500x300")

label_pytanie = tk.Label(root, text="Tutaj będzie pytanie", font=("Arial", 14), wraplength=480, justify="left")
label_pytanie.pack(pady=20)

var = tk.IntVar()
radio = []
for i in range(3):
  r = tk.Radiobutton(root, text="", variable=var, value=i, font=("Arial", 12), anchor="w", justify="left")
  r.pack(fill="x", padx=40)
  radio.append(r)

btn = tk.Button(root, text="Dalej", command=dalej)
btn.pack(pady=20)

# Pokaż pytanie na starcie
wyswietl_pytanie()
root.mainloop()
```





Zdjęcie jako tło

```
import tkinter as tk
from PIL import Image, ImageTk

root = tk.Tk()
root.title("Formularz z tłem")
root.geometry("800x600")

bg_image = Image.open("zdjecie1.jpg").resize((800, 600))
bg_photo = ImageTk.PhotoImage(bg_image)

canvas = tk.Canvas(root, width=800, height=600)
canvas.pack(fill="both", expand=True)
canvas.create_image(0, 0, image=bg_photo, anchor="nw")

def create_field(label_text, row_y):
canvas.create_text(150, row_y, text=label_text, fill="black",
font=("Arial", 18), anchor="w")
entry = tk.Entry(root, font=("Arial", 14))
entry_window = canvas.create_window(300, row_y,
window=entry, anchor="w", width=250)
return entry

entry_name = create_field("Imię:", 100)
entry_surname = create_field("Nazwisko:", 150)
entry_email = create_field("Email:", 200)
entry_age = create_field("Wiek:", 250)


def submit():
print("Imię:", entry_name.get())
print("Nazwisko:", entry_surname.get())
print("Email:", entry_email.get())
print("Wiek:", entry_age.get())

submit_btn = tk.Button(root, text="Wyślij", font=("Arial", 14),
command=submit)
canvas.create_window(300, 320, window=submit_btn,
anchor="w")

root.mainloop()
```

Wygląd i działanie

To prosta aplikacja w formie formularza, w której użytkownik wprowadza i zapisuje dane w pamięci. Canvas w najprostszej postaci pozwala na proste umieszczenie tła w całym obszarze okna lub na wybranym panelu.

Gradienty jako tła

```
import tkinter as tk

from PIL import Image, ImageTk, ImageDraw


def create_gradient(width, height, color1, color2):

base = Image.new('RGB', (width, height), color1)

top = Image.new('RGB', (width, height), color2)

mask = Image.new('L', (width, height))


draw = ImageDraw.Draw(mask)

for y in range(height):

draw.line([(0, y), (width, y)], fill=int(255 * (y / height)))

base.paste(top, (0, 0), mask)

return base


root = tk.Tk()

root.title("Formularz z gradientem")

root.geometry("800x600")


gradient_img = create_gradient(800, 600, "#003366", "#66ccff")

gradient_photo = ImageTk.PhotoImage(gradient_img)


canvas = tk.Canvas(root, width=800, height=600)

canvas.pack(fill="both", expand=True)

canvas.create_image(0, 0, image=gradient_photo, anchor="nw")


def create_field(label_text, row_y):

canvas.create_text(150, row_y, text=label_text, fill="white", font=("Arial", 14), anchor="w")

entry = tk.Entry(root, font=("Arial", 14))

canvas.create_window(300, row_y, window=entry, anchor="w", width=250)

return entry


entry_name = create_field("Imię:", 100)

entry_surname = create_field("Nazwisko:", 150)

entry_email = create_field("Email:", 200)

entry_age = create_field("Wiek:", 250)


def submit():

print("Imię:", entry_name.get())

print("Nazwisko:", entry_surname.get())

print("Email:", entry_email.get())

print("Wiek:", entry_age.get())


submit_btn = tk.Button(root, text="Wyślij", font=("Arial", 14), command=submit)

canvas.create_window(300, 320, window=submit_btn, anchor="w")


root.mainloop()
```

Wygląd...

Zdarzenia myszy w Tkinter

Zdarzenie	Opis	Przykład zapisu bind
<Button-1>	Kliknięcie lewym przyciskiem myszy	widget.bind("<Button-1>", handler)
<Button-2>	Kliknięcie środkowym przyciskiem (kółkiem) myszy	widget.bind("<Button-2>", handler)
<Button-3>	Kliknięcie prawym przyciskiem myszy	widget.bind("<Button-3>", handler)
<Double-Button-1>	Podwójne kliknięcie lewym przyciskiem	widget.bind("<Double-Button-1>", handler)
<Triple-Button-1>	Potrójne kliknięcie lewym przyciskiem	widget.bind("<Triple-Button-1>", handler)
<B1-Motion>	Przeciąganie z wciśniętym lewym przyciskiem	widget.bind("<B1-Motion>", handler)
<Enter>	Wjechanie kursorem na widget	widget.bind("<Enter>", handler)
<Leave>	Wyjechanie kursorem z widgetu	widget.bind("<Leave>", handler)
<Motion>	Ruch kursora myszy nad widgetem	widget.bind("<Motion>", handler)
<ButtonRelease-1>	Zwolnienie lewego przycisku myszy	widget.bind("<ButtonRelease-1>", handler)
<ButtonRelease-2>	Zwolnienie środkowego przycisku	widget.bind("<ButtonRelease-2>", handler)
<ButtonRelease-3>	Zwolnienie prawego przycisku	widget.bind("<ButtonRelease-3>", handler)
<MouseWheel>	Scroll kółkiem myszy (Windows)	widget.bind("<MouseWheel>", handler)
<Button-4>	Scroll kółkiem myszy w górę (Linux)	widget.bind("<Button-4>", handler)
<Button-5>	Scroll kółkiem myszy w dół (Linux)	widget.bind("<Button-5>", handler)

Zdarzenia myszy w Tkinter

```
def on_click(event):
    print(f"Kliknięto w pozycji: {event.x}, {event.y}")
widget.bind("<Button-1>", on_click)
```

W obiekcie event masz m.in.:

- event.x, event.y – współrzędne kliknięcia względem widgetu,
- event.widget – widget, który wywołał zdarzenie,
- event.num – numer przycisku myszy (1-lewy, 2-środkowy, 3-prawy).

Przykład

```
import tkinter as tk


def on_click(event):
    info = f"Kliknięto przyciskiem {event.num} w pozycji ({event.x}, {event.y})"
    print(info)
    label.config(text=info)


def on_double_click(event):
    info = f"Podwójne kliknięcie przyciskiem {event.num} w pozycji ({event.x}, {event.y})"
    print(info)
    label.config(text=info)


def on_motion(event):
    info = f"Ruch myszy na pozycji ({event.x}, {event.y})"
    label.config(text=info)


def on_enter(event):
    label.config(text="Kursor wszedł na widget")


def on_leave(event):
    label.config(text="Kursor opuścił widget")


root = tk.Tk()
root.title("Obsługa zdarzeń myszy")

frame = tk.Frame(root, width=400, height=300, bg="lightgray")
frame.pack(padx=20, pady=20)

label = tk.Label(root, text="Interakcja z myszą...", font=("Arial", 14))
label.pack(pady=10)

# Podpinamy zdarzenia do ramki
frame.bind("<Button-1>", on_click)       # Lewy przycisk kliknięcie
frame.bind("<Button-2>", on_click)       # Środkowy przycisk kliknięcie
frame.bind("<Button-3>", on_click)       # Prawy przycisk kliknięcie
frame.bind("<Double-Button-1>", on_double_click) # Podwójne kliknięcie lewym przyciskiem
frame.bind("<Motion>", on_motion)        # Ruch myszy
frame.bind("<Enter>", on_enter)          # Wejście kursora na widget
frame.bind("<Leave>", on_leave)          # Wyjście kursora z widgetu

root.mainloop()
```

Efekt działania programu

W zależności od sposobu zachowania myszy program wskazuje pozycję oraz rozpoznanie jaki przycisk i ile razy został wciśnięty.

Zdarzenia klawiatury w Tkinter

Zdarzenie	Opis	Przykład bind
<KeyPress> lub <Key>	Naciśnięcie dowolnego klawisza	widget.bind("<KeyPress>", handler)
<KeyRelease>	Zwolnienie dowolnego klawisza	widget.bind("<KeyRelease>", handler)
<KeyPress-a>	Naciśnięcie konkretnego klawisza, np. 'a'	widget.bind("<KeyPress-a>", handler)
<KeyRelease-a>	Zwolnienie konkretnego klawisza	widget.bind("<KeyRelease-a>", handler)
<Return>	Naciśnięcie klawisza Enter (Return)	widget.bind("<Return>", handler)
<BackSpace>	Naciśnięcie Backspace	widget.bind("<BackSpace>", handler)
<Tab>	Naciśnięcie klawisza Tab	widget.bind("<Tab>", handler)
<Escape>	Naciśnięcie klawisza Esc	widget.bind("<Escape>", handler)
<Up>, <Down>, <Left>, <Right>	Strzałki kierunkowe	widget.bind("<Up>", handler) i analogicznie

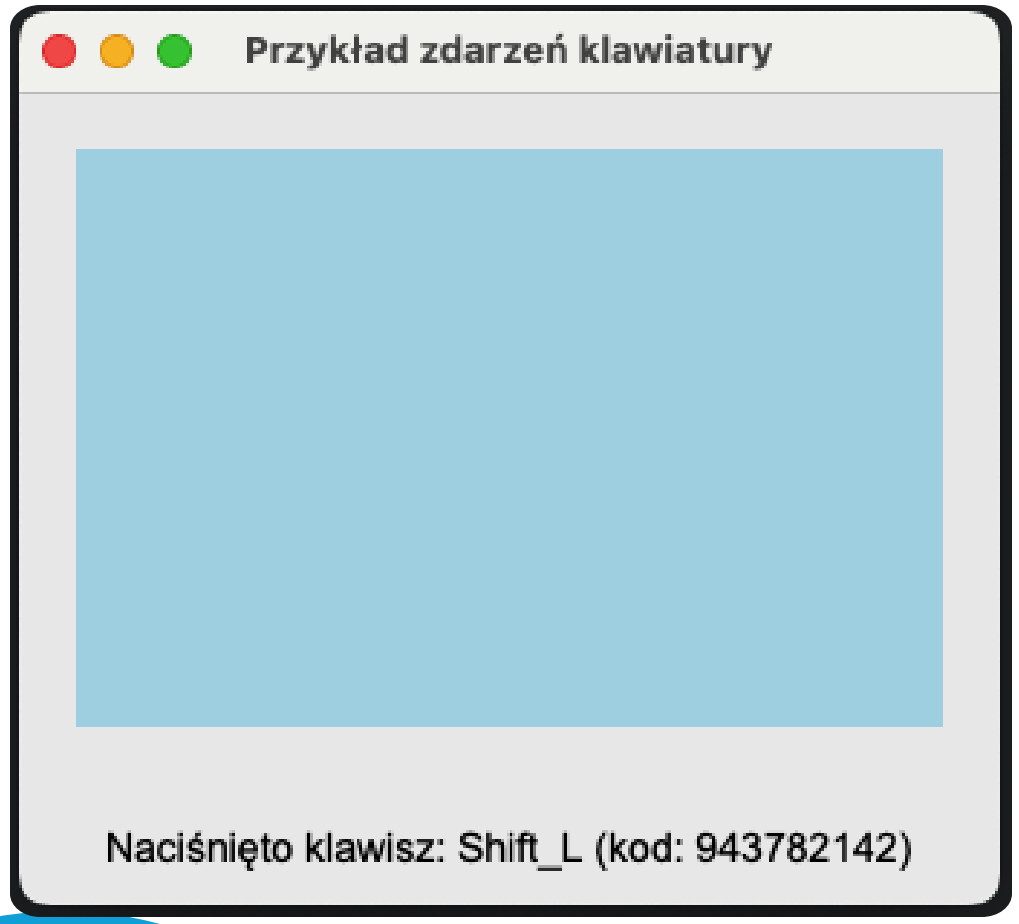
Przykład

```
def on_key(event):
    print(f"Naciśnięto klawisz: {event.keysym}, kod: {event.keycode}")

widget.bind("<KeyPress>", on_key)
```

Kilka uwag:

- event.keysym — czytelna nazwa klawisza (np. 'a', 'Return', 'Escape').
- event.keycode — numer kodu klawisza (systemowy).
- Można również obsługiwać konkretne klawisze (np. <KeyPress-a>) zamiast wszystkich.

Zdarzenia pod wybrane przyciski




```
import tkinter as tk

def on_key(event):
    info = f"Naciśnięto klawisz: {event.keysym} (kod: {event.keycode})"
    print(info)
    label.config(text=info)

def on_enter(event):
    info = "Naciśnięto Enter"
    print(info)
    label.config(text=info)

def on_escape(event):
    info = "Naciśnięto Escape"
    print(info)
    label.config(text=info)

def on_up(event):
    info = "Naciśnięto strzałkę w górę"
    print(info)
    label.config(text=info)

root = tk.Tk()
root.title("Przykład zdarzeń klawiatury")

frame = tk.Frame(root, width=300, height=200, bg="lightblue")
frame.pack(padx=20, pady=20)

label = tk.Label(root, text="Naciśnij klawisz...", font=("Arial", 14))
label.pack(pady=10)

frame.focus_set()

# Bindujemy różne zdarzenia klawiaturowe
frame.bind("<KeyPress>", on_key)       # dowolny klawisz
frame.bind("<Return>", on_enter)       # Enter
frame.bind("<Escape>", on_escape)      # Escape
frame.bind("<Up>", on_up)              # strzałka w górę

root.mainloop()
```

Projektowanie GUI

- Tworząc aplikację graficzną, bardzo ważne jest **ergonomiczne i estetyczne** rozmieszczenie elementów (widgetów), tak aby interfejs był:
- **Intuicyjny** i łatwy w użyciu
- **Estetyczny** i spójny wizualnie
- **Responsywny** – dobrze wyglądał przy różnych rozmiarach okna
- Tkinter oferuje **3 główne mechanizmy układu (layout managers)**:

Pack() – układ liniowy

Charakterystyka:

- Prosty w użyciu
- Umieszcza widgety **jeden po drugim**: pionowo lub poziomo
- Automatycznie dopasowuje rozmiar.

```
import tkinter as tk

root = tk.Tk()

label1 = tk.Label(root, text="Góra", bg="lightblue")
label1.pack(side="top", fill="x")

label2 = tk.Label(root, text="Dół", bg="lightgreen")
label2.pack(side="bottom", fill="x")

root.mainloop()
```

Pack() – układ liniowy

Opcje:

- side: "top", "bottom", "lef t", "right"
- fill: "x", "y", "both"
- expand: rozciąga widget w dostępnej przestrzeni
- padx, pady: odstępy zewnętrzne.

Grid() – układ siatkowy

Charakterystyka:

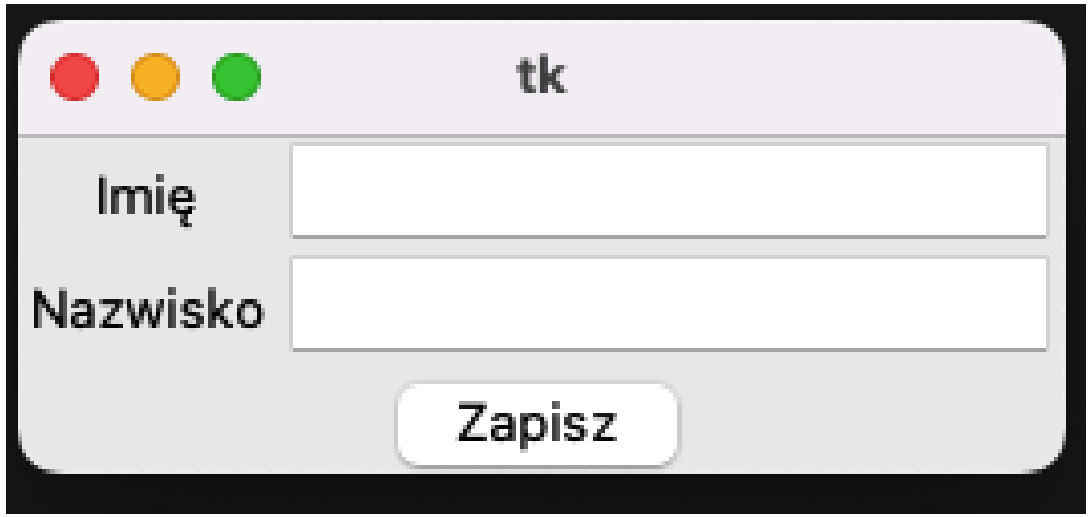
- Układ podobny do **tabeli**
- Umożliwia rozmieszczanie widgetów w **wierszach i kolumnach**
- Precyzyjna kontrola nad pozycją

```
import tkinter as tk

root = tk.Tk()

tk.Label(root, text="Imię").grid(row=0, column=0)
tk.Entry(root).grid(row=0, column=1)

tk.Label(root, text="Nazwisko").grid(row=1, column=0)
tk.Entry(root).grid(row=1, column=1)

tk.Button(root, text="Zapisz").grid(row=2, column=0, columnspan=2)

root.mainloop()
```

Grid() – układ siatkowy

Opcje:

- row, column: pozycja w siatce
- rowspan, columnspan: zajmowanie kilku komórek
- sticky: przypięcie do kierunku (n, s, e, w)
- padx, pady.



Place() – układ absolutny

Charakterystyka:

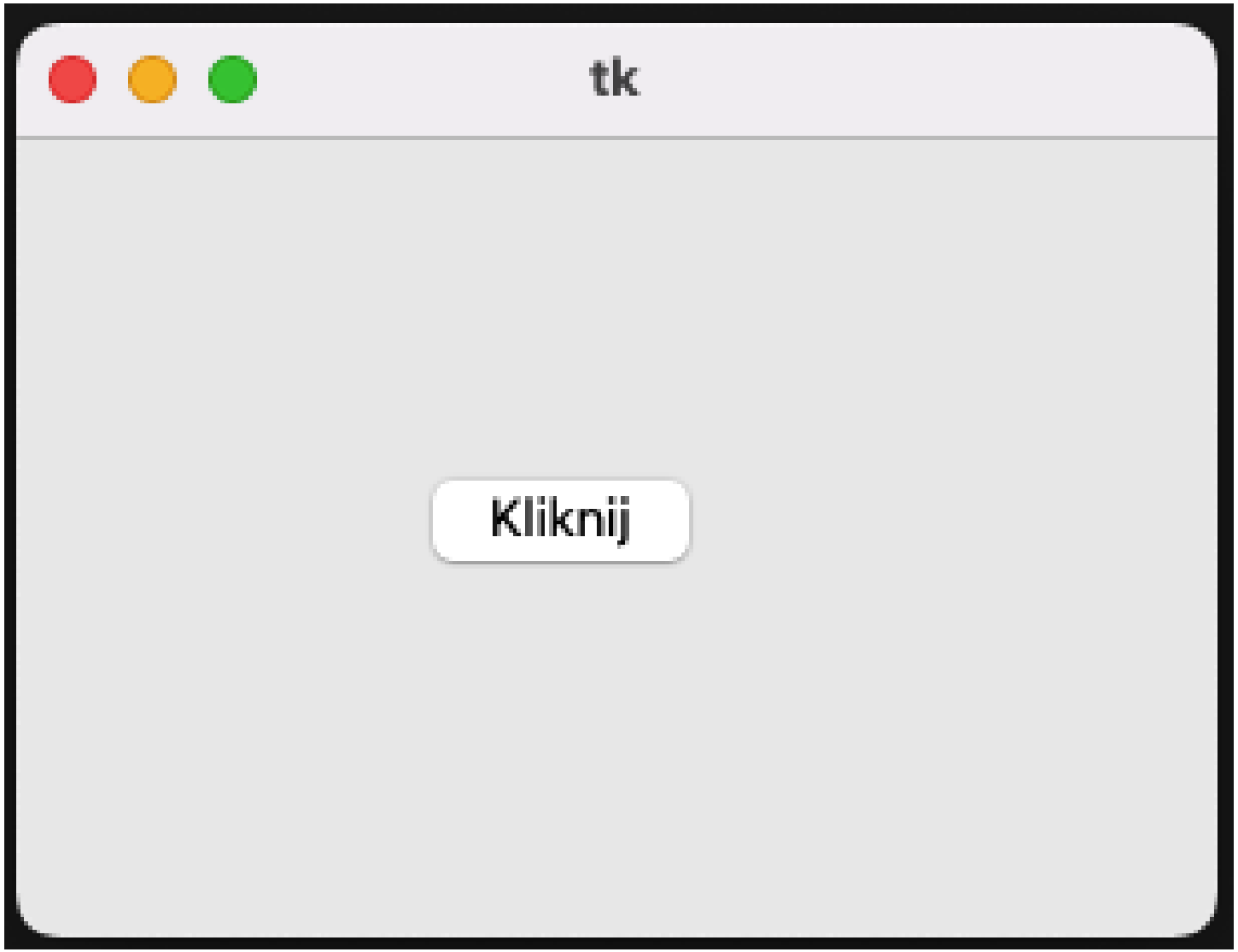
- Pozwala na **dokładne określenie pozycji i rozmiaru**
- Wymaga ręcznego ustawiania współrzędnych
- Najmniej elastyczny – nieprzystosowany do skalowania.

```
import tkinter as tk

root = tk.Tk()
root.geometry("300x200")

button = tk.Button(root, text="Kliknij")
button.place(x=100, y=80)

root.mainloop()
```

Place() – układ absolutny

Opcje:

- x, y: współrzędne
- width, height: wymiary
- relx, rely: względne pozycjonowanie (0.0–1.0).



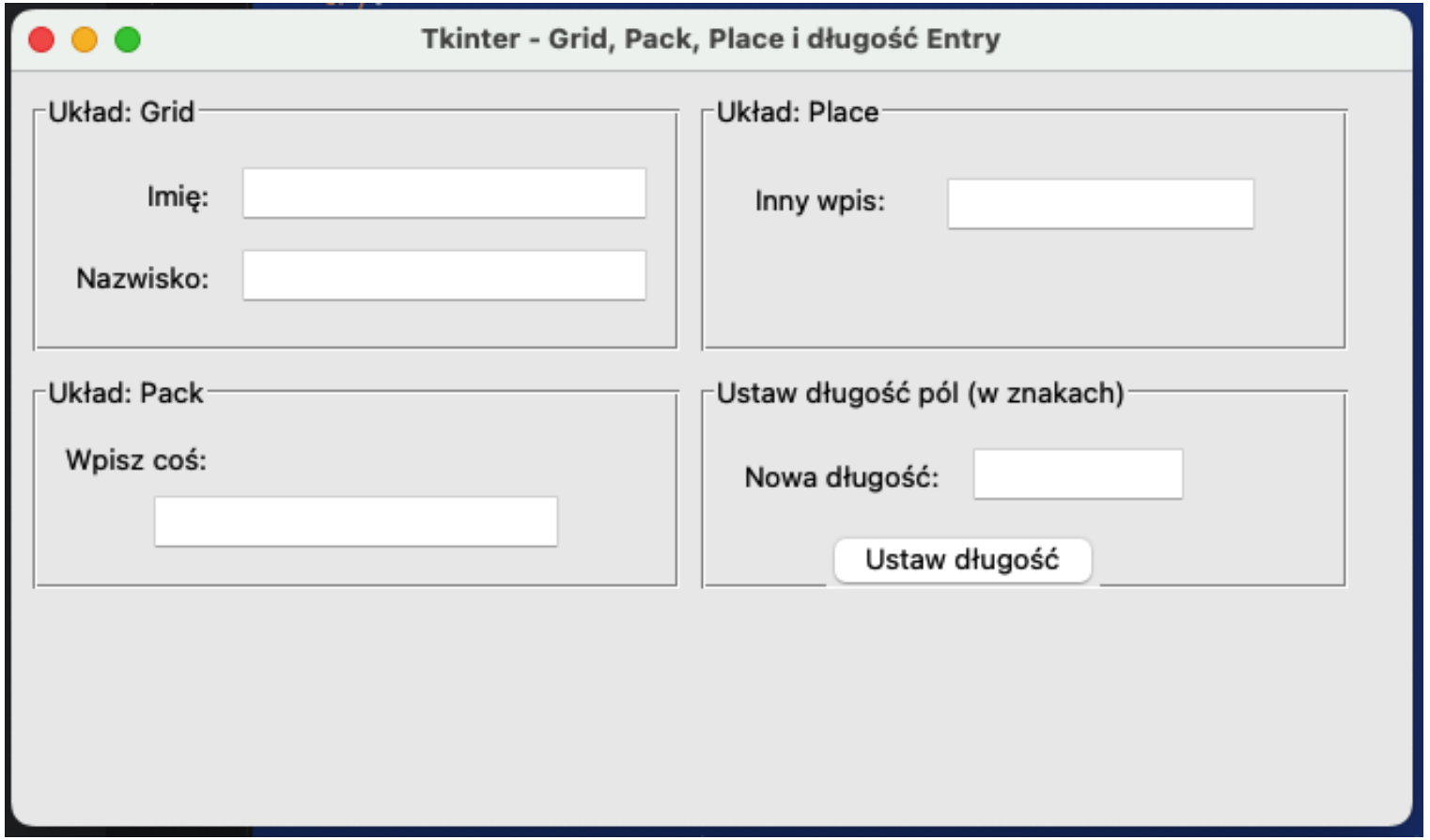
Wybór layout …

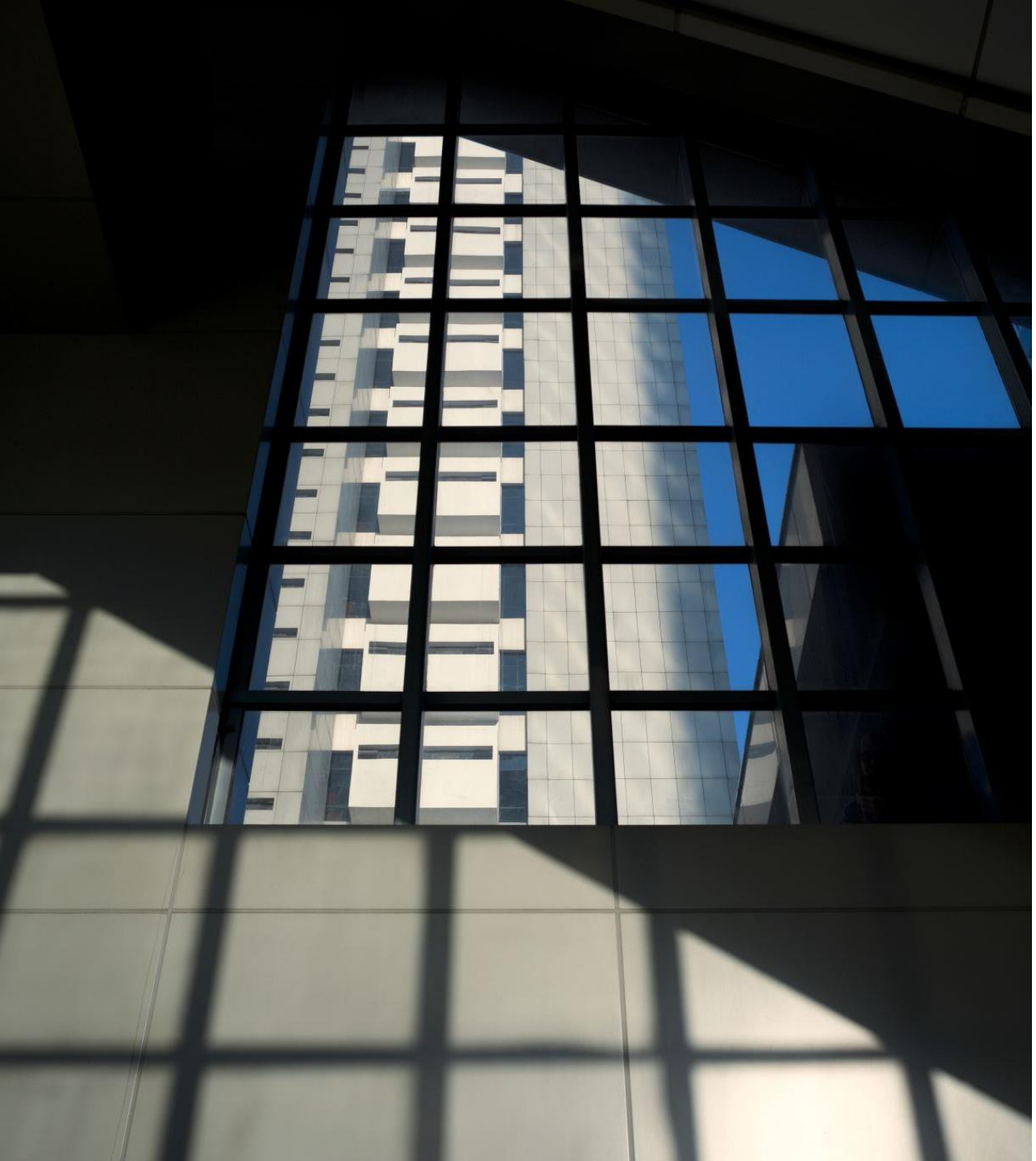
Layout	Plusy	Minusy	Zastosowanie
pack()	Prosty, szybki	Mała kontrola nad pozycją	Proste interfejsy
grid()	Elastyczny, logiczny	Nie działa razem z pack()	Formularze, układy tabelaryczne
place()	Precyzyjny	Słabo się skaluje	Układy specjalne, grafika

Grid, Pack i Place w jednym oknie

```
import tkinter as tk

def ustaw_dlugosc():
    try:
        nowa_dlugosc = int(entry_dlugosc.get())
        if nowa_dlugosc <= 0:
            raise ValueError
        # Zmiana długości entry w różnych układach
        entry_grid_imie.config(width=nowa_dlugosc)
        entry_grid_nazwisko.config(width=nowa_dlugosc)
        entry_pack.config(width=nowa_dlugosc)
        entry_place.config(width=nowa_dlugosc)
        label_info.config(text="Długość pól zmieniona", fg="green")
    except ValueError:
        label_info.config(text="Podaj poprawną liczbę całkowitą > 0", fg="red")

root = tk.Tk()
root.title("Tkinter - Grid, Pack, Place i długość Entry")
root.geometry("650x350")

# ---------- GRID ----------
frame_grid = tk.LabelFrame(root, text="Układ: Grid", padx=10, pady=10)
frame_grid.place(x=10, y=10, width=300, height=120)

tk.Label(frame_grid, text="Imię:").grid(row=0, column=0, sticky="e", padx=5, pady=5)
entry_grid_imie = tk.Entry(frame_grid, width=20)
entry_grid_imie.grid(row=0, column=1, padx=(5, 20), pady=5)

tk.Label(frame_grid, text="Nazwisko:").grid(row=1, column=0, sticky="e", padx=5, pady=5)
entry_grid_nazwisko = tk.Entry(frame_grid, width=20)
entry_grid_nazwisko.grid(row=1, column=1, padx=(5, 20), pady=5)

# ---------- PACK ----------
frame_pack = tk.LabelFrame(root, text="Układ: Pack", padx=10, pady=10)
frame_pack.place(x=10, y=140, width=300, height=100)

tk.Label(frame_pack, text="Wpisz coś:").pack(anchor="w")
entry_pack = tk.Entry(frame_pack, width=20)
entry_pack.pack(pady=5)

# ---------- PLACE ----------
frame_place = tk.LabelFrame(root, text="Układ: Place", padx=10, pady=10)
frame_place.place(x=320, y=10, width=300, height=120)

tk.Label(frame_place, text="Inny wpis:").place(x=10, y=10)
entry_place = tk.Entry(frame_place, width=15)
entry_place.place(x=100, y=10)

# ---------- Ustawianie długości ----------
frame_dlugosc = tk.LabelFrame(root, text="Ustaw długość pól (w znakach)", padx=10,
pady=10)
frame_dlugosc.place(x=320, y=140, width=300, height=100)

tk.Label(frame_dlugosc, text="Nowa długość:").grid(row=0, column=0, sticky="e", padx=5,
pady=5)
entry_dlugosc = tk.Entry(frame_dlugosc, width=10)
entry_dlugosc.grid(row=0, column=1, padx=5, pady=5)

btn_ustaw = tk.Button(frame_dlugosc, text="Ustaw długość", command=ustaw_dlugosc)
btn_ustaw.grid(row=1, column=0, columnspan=2, pady=5)

# ---------- Info ----------
label_info = tk.Label(root, text="")
label_info.place(x=10, y=260)

root.mainloop()
```

Efekt działania programu





Dobre praktyki

- **Używaj grid() do formularzy**, pack() do ogólnej struktury.
- **Używaj Frame do grupowania widgetów** i mieszania layoutów.
- **Unikaj place()**, jeśli nie musisz – trudniejszy w utrzymaniu.
- **Testuj skalowalność GUI** – czy wygląda dobrze przy różnych rozmiarach okna?
- **Zadbaj o marginesy i wyrównanie** (padx, pady, sticky).

Canvas

```
canvas = tk.Canvas(parent, width=width, height=height, bg="color")

canvas.pack()
```

Metoda	Opis	Przykład
create_line(x1, y1, x2, y2, ...)	Rysuje linię między punktami	canvas.create_line(0,0,100,100)
create_rectangle(x1, y1, x2, y2, ...)	Rysuje prostokąt (x1,y1) i (x2,y2)	canvas.create_rectangle(10,10,50,50)
create_oval(x1, y1, x2, y2, ...)	Rysuje elipsę wpisaną w prostokąt	canvas.create_oval(10,10,50,50)
create_text(x, y, text=..., ...)	Dodaje tekst w punkcie (x,y)	canvas.create_text(100,100, text="Hej!")
create_polygon(coords, ...)	Rysuje wielokąt na podstawie listy punktów	canvas.create_polygon(10,10, 50,10, 30,50)
create_image(x, y, image=...)	Umieszcza obraz w punkcie (x,y)	canvas.create_image(0,0, image=img, anchor='nw')

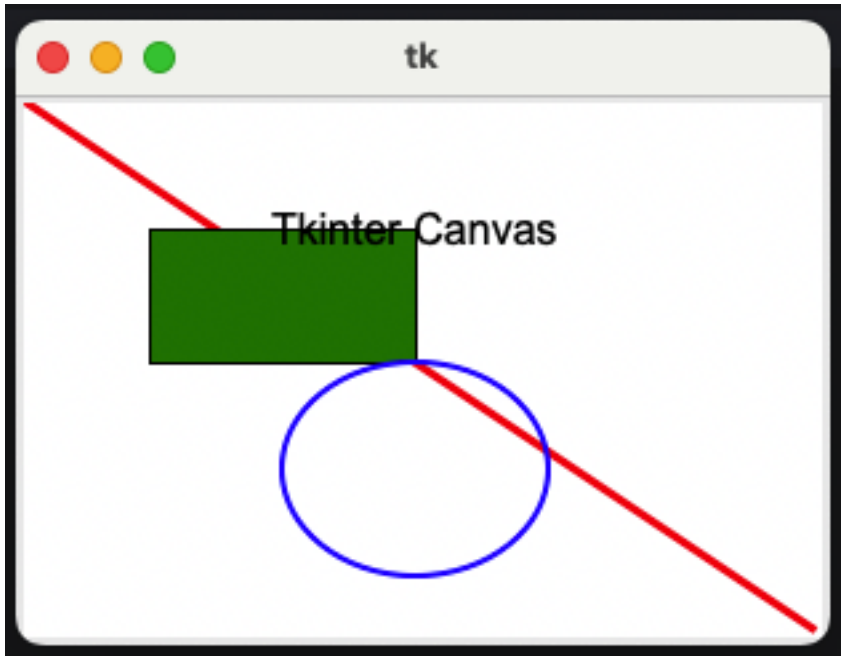
Przykłady zastosowania

```
rect_id = canvas.create_rectangle(10,10,60,60, fill="red")
canvas.move(rect_id, 20, 0)          # przesunięcie o 20 px w prawo
canvas.itemconfig(rect_id, fill="blue")  # zmiana koloru na niebieski
```

```
canvas.delete(rect_id)  # usuwa konkretny obiekt
canvas.delete("all")    # usuwa wszystko z canvas
```

```
import tkinter as tk

root = tk.Tk()
canvas = tk.Canvas(root, width=300, height=200, bg="white")
canvas.pack()

canvas.create_line(0, 0, 300, 200, fill="red", width=3)
canvas.create_rectangle(50, 50, 150, 100, fill="green")
canvas.create_oval(100, 100, 200, 180, outline="blue", width=2)
canvas.create_text(150, 50, text="Tkinter Canvas", font=("Arial",
16))

root.mainloop()
```

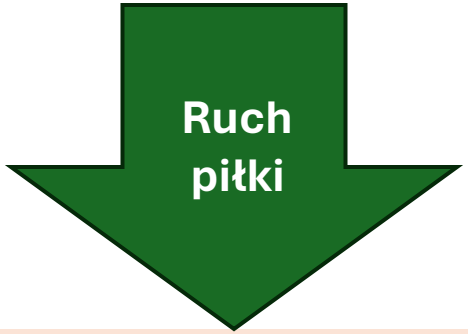
Efekt?



Różne figury w oknie. Co można z tym zrobić? W dowolny kształt możemy wpisać widżety oraz dopisać do nich funkcje. Canvas będzie potrzebny również do przedstawiania danych na wykresach lub dzielenia widoku aplikacji na sekcje. Zupełnie jak podczas projektowania witryn internetowych.

Canvas i animacje

```
def animate():
    canvas.move(object_id, 5, 0)  # przesuwa obiekt o 5 px w prawo
    root.after(30, animate)      # wykonuj co 30 ms
```



Możesz animować:

- **położenie (canvas.move, canvas.coords)**
- **kolor (canvas.itemconfig)**
- **tekst (canvas.itemconfig)**
- **skalowanie (np. przez zmianę współrzędnych lub scale)**
- **przezroczystość (trudniej – trzeba obejść).**

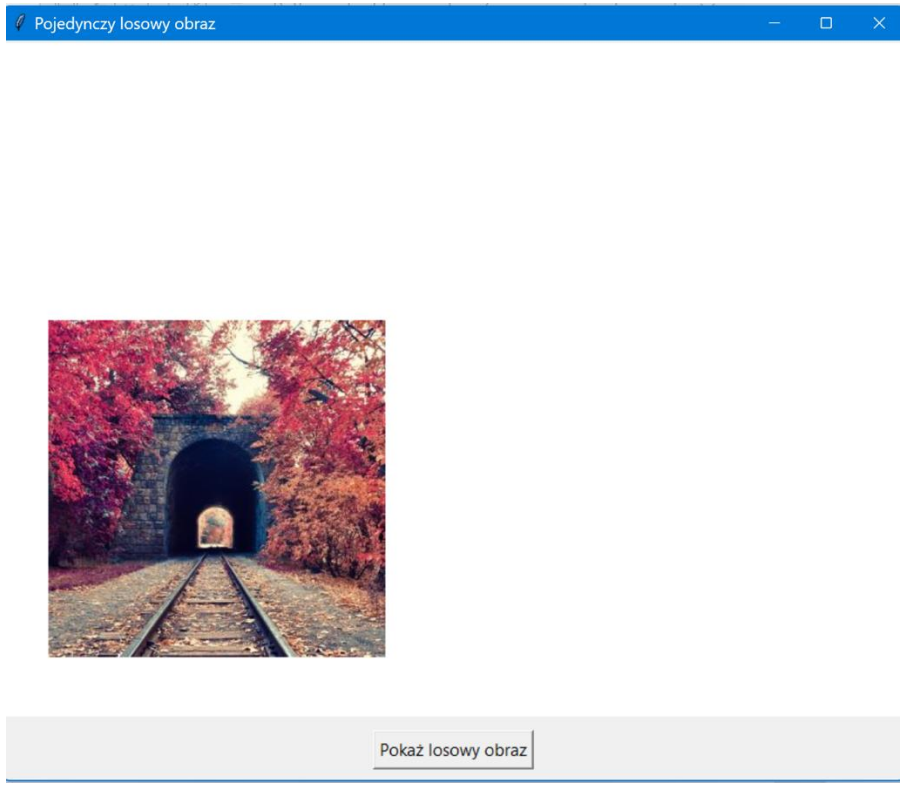
```
import tkinter as tk

root = tk.Tk()
canvas = tk.Canvas(root, width=400, height=300, bg="white")
canvas.pack()

ball = canvas.create_oval(10, 10, 50, 50, fill="blue")
dx = 5
dy = 3

def move_ball():
    nonlocal dx, dy
    canvas.move(ball, dx, dy)
    x1, y1, x2, y2 = canvas.coords(ball)

    if x1 <= 0 or x2 >= 400:
        dx = -dx
    if y1 <= 0 or y2 >= 300:
        dy = -dy

    root.after(20, move_ball)

move_ball()
root.mainloop()
```

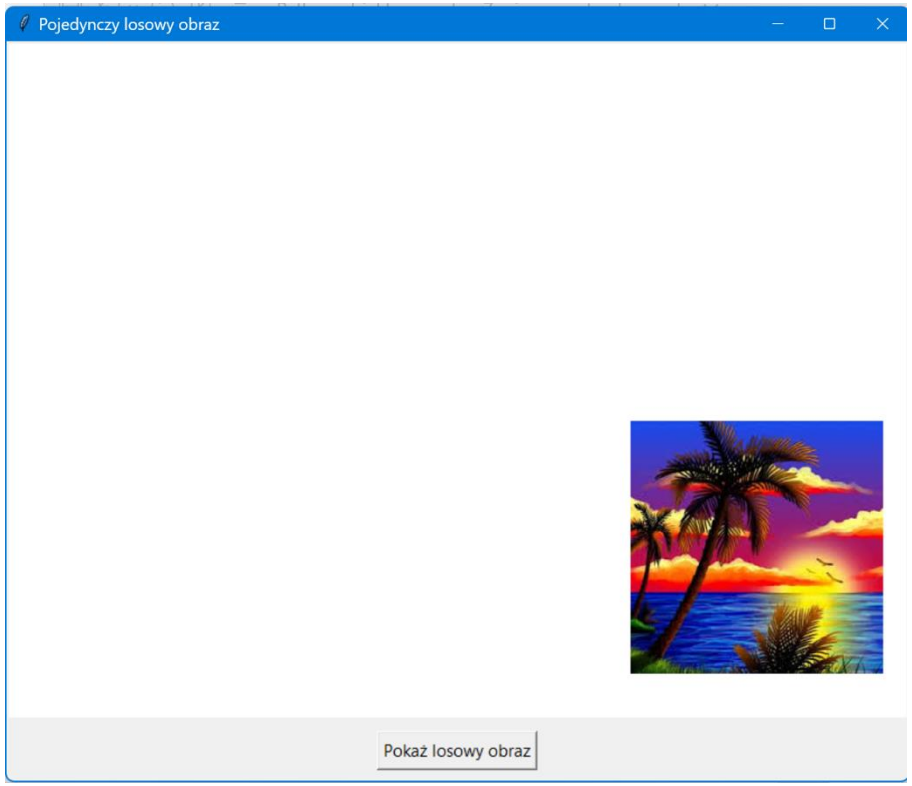
Przykład – animacja emoji

```
import tkinter as tk

root = tk.Tk()
root.title("Animacja emoji")

canvas = tk.Canvas(root, width=200, height=100, bg="white")
canvas.pack()

# Lista "klatek" animacji emoji
frames = ["🚶", "🚶‍♂️", "🚶", "🚶‍♂️"]

current_frame = 0
emoji_text = canvas.create_text(100, 50, text=frames[current_frame], font=("Arial", 48))

def animate():
    global current_frame
    current_frame = (current_frame + 1) % len(frames)
    canvas.itemconfig(emoji_text, text=frames[current_frame])
    root.after(200, animate)

animate()
root.mainloop()
```

Random na obiektach

```
import tkinter as tk
from PIL import Image, ImageTk
import random
import os

root = tk.Tk()
root.title("Pojedynczy losowy obraz")

canvas = tk.Canvas(root, width=800,
height=600, bg="white")
canvas.pack()

# Ścieżka do folderu ze zdjęciami
sciezka_folderu = "zdjecia"

# Lista dostępnych plików graficznych
dostepne_pliki = [f for f in
os.listdir(sciezka_folderu)
                  if f.lower().endswith(('.jpg', '.jpeg',
'.png', '.gif'))]

if not dostepne_pliki:
    print("Brak obrazów w folderze 'zdjecia'.")
    exit(1)

# Lista referencji do zdjęć (aby ich nie usunęło z
pamięci)
photos = []

def pokaz_jeden_obraz():
    canvas.delete("all")  # Czyść poprzednie
obrazy

    losowy_plik = random.choice(dostepne_pliki)
    sciezka = os.path.join(sciezka_folderu,
losowy_plik)

    try:
        obraz = Image.open(sciezka)
        img_width, img_height = obraz.size
        x = random.randint(0, 800 - img_width)
        y = random.randint(0, 600 - img_height)
        photo = ImageTk.PhotoImage(obraz)
        canvas.create_image(x, y, image=photo,
anchor="nw")
        photos.clear()
        photos.append(photo)  # Przechowujemy
tylko ten jeden obraz
    except Exception as e:
        print(f"Błąd ładowania obrazu
{losowy_plik}:", e)

# Przycisk do wyświetlania jednego losowego
obrazu
button = tk.Button(root, text="Pokaż losowy
obraz", command=pokaz_jeden_obraz)
button.pack(pady=10)

root.mainloop()
```





Efekt działania programu

Opis działania programu

Program losuje z listy zdjęcia a następnie wyświetla przypadkowe z nich. Konieczne jest dołączenie obrazków umieszczonych w folderze. Program wykorzystuje Canvas do umieszczania zdjęć wewnątrz okna.

Gra w „Kości”

```
import tkinter as tk
from tkinter import messagebox
from PIL import Image, ImageTk
import random
import os

class DiceGame:
    def __init__(self, root):
        self.root = root
        self.root.title("Gra w Kości")
        self.root.resizable(False, False)

        # Wczytaj obrazy kostek
        self.dice_images = []
        for i in range(1, 7):
            img_path = f"dice{i}.png"
            if not os.path.exists(img_path):
                messagebox.showerror("Błąd", f"Nie znaleziono pliku: {img_path}")
                self.root.destroy()
                return
            image = Image.open(img_path).resize((100, 100))
            self.dice_images.append(ImageTk.PhotoImage(image))

        # Utwórz ramkę na kości
        self.dice_frame = tk.Frame(self.root)
        self.dice_frame.pack(pady=20)

        # Etykiety dla pięciu kości
        self.dice_labels = []
        for _ in range(5):
            label = tk.Label(self.dice_frame, image=self.dice_images[0])
            label.pack(side=tk.LEFT, padx=5)
            self.dice_labels.append(label)

        # Przycisk rzutu
        self.roll_button = tk.Button(self.root, text="Rzuć kośćmi", font=("Arial", 14),
command=self.roll_dice)
        self.roll_button.pack(pady=10)

        # Wynik tekstowy
        self.result_label = tk.Label(self.root, text="", font=("Arial", 14))
        self.result_label.pack(pady=10)

    def roll_dice(self):
        results = [random.randint(1, 6) for _ in range(5)]
        for i, value in enumerate(results):
            self.dice_labels[i].configure(image=self.dice_images[value - 1])

        self.result_label.config(text="Wyniki: " + " ".join(str(r) for r in results))

if __name__ == "__main__":
    root = tk.Tk()
    game = DiceGame(root)
    root.mainloop()
```

Wygląd gry i opis działania

Gra uruchamia się za pomocą przycisku. Losowanych jest 5 wyników. Do poprawnego działania gry potrzebne są zdjęcia.

Gra nie wymaga bibliotek typowych dla gier. Nie ma w niej ruchu postaci.

Podsumowanie - pytania

- Czym jest programowanie obiektowe?
- Wyjaśnij pojęcia polimorfizmu i hermetyzacji.
- Wymień przynajmniej 5 bibliotek Pythona i wyjaśnij ich działanie.
- Do czego służą Django i Flask?
- Podaj najważniejsze cechy Tkinter?
- Wymień przynajmniej 5 widżetów Tkinter.
- Wymień nazwy podstawowych układów w Tkinter.

Zadania do samodzielnego wykonania

Zadanie 1

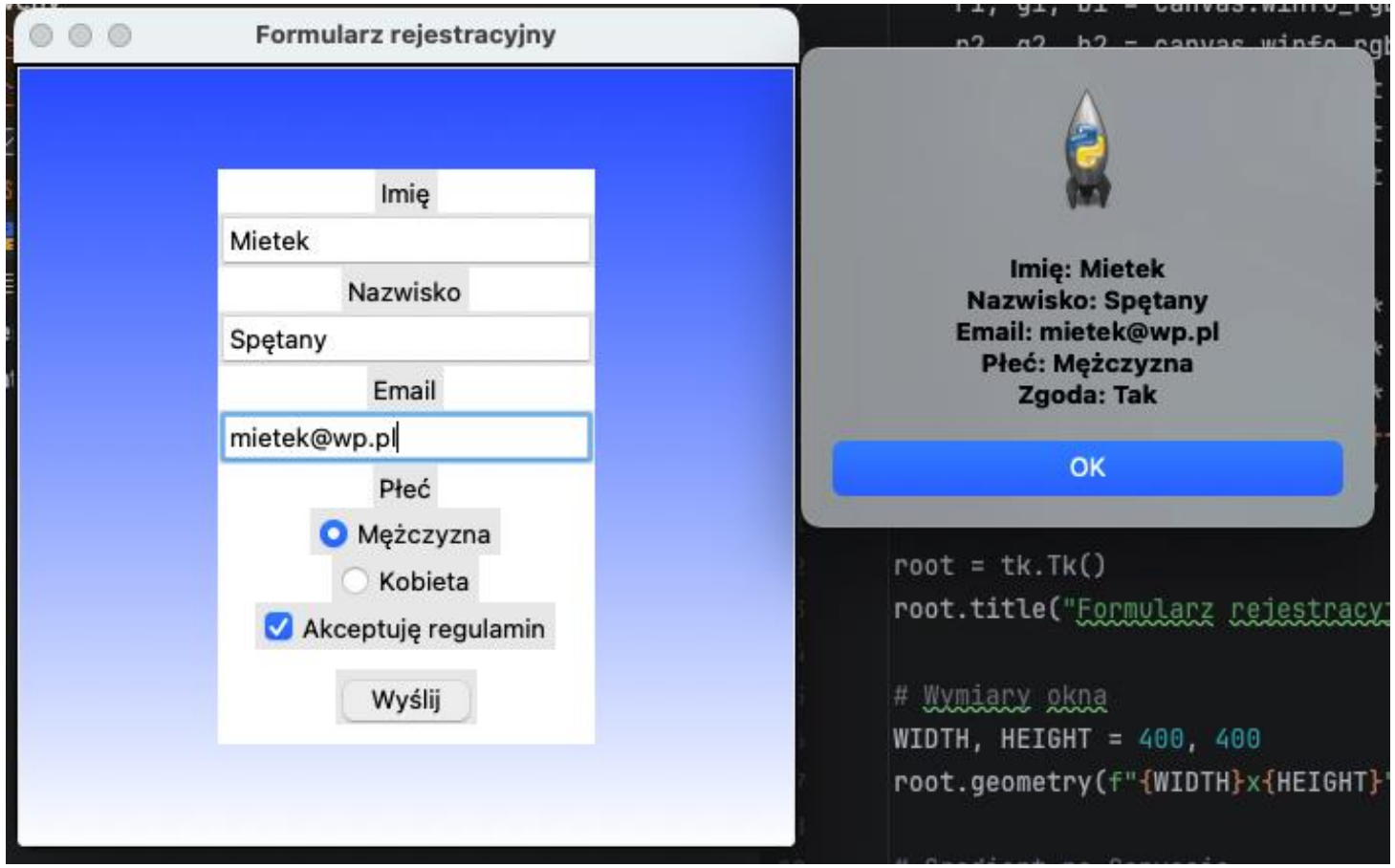
Kalkulator - wprowadzasz dwie liczby, wybierasz działanie i wynik zostaje wyświetlony na etykiecie. Kalkulator korzysta z GRID, zmiennych typu float oraz dopisania funkcji do przycisku przez lambdę.

Zadania do samodzielnego wykonania

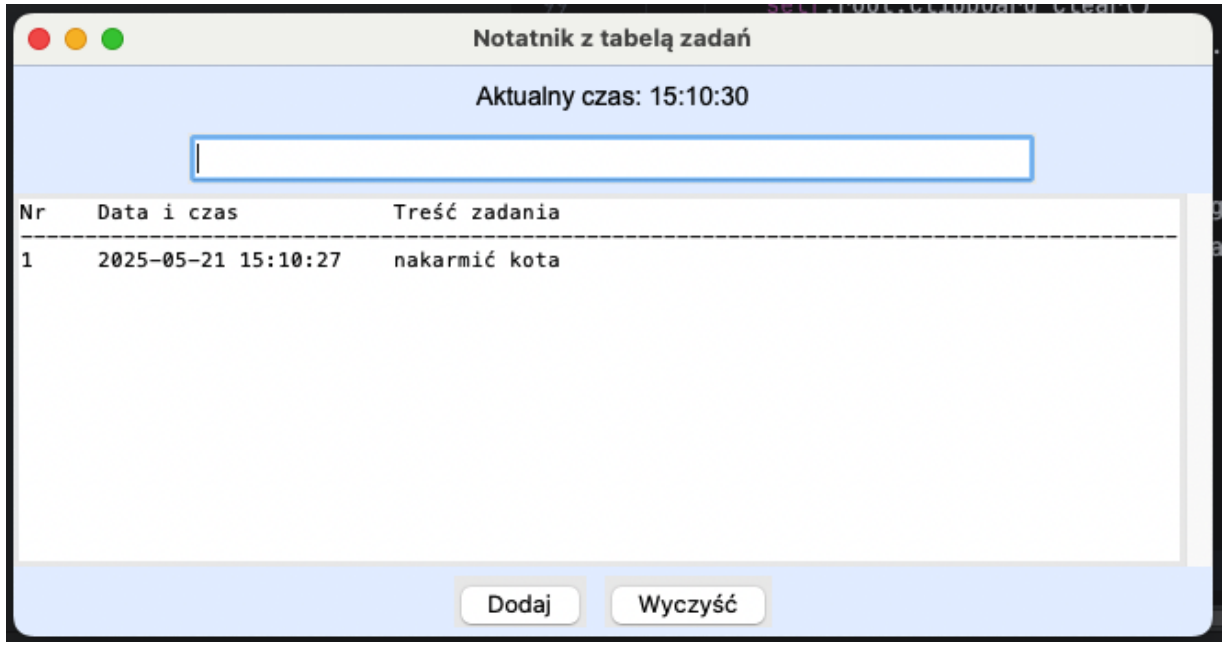
Zadanie 2

Formularz rejestracyjny – dane wprowadza użytkownik, następnie są wyświetlane w wyskakującym okienku.

Do formatowania użyto CANVAS i gradient.

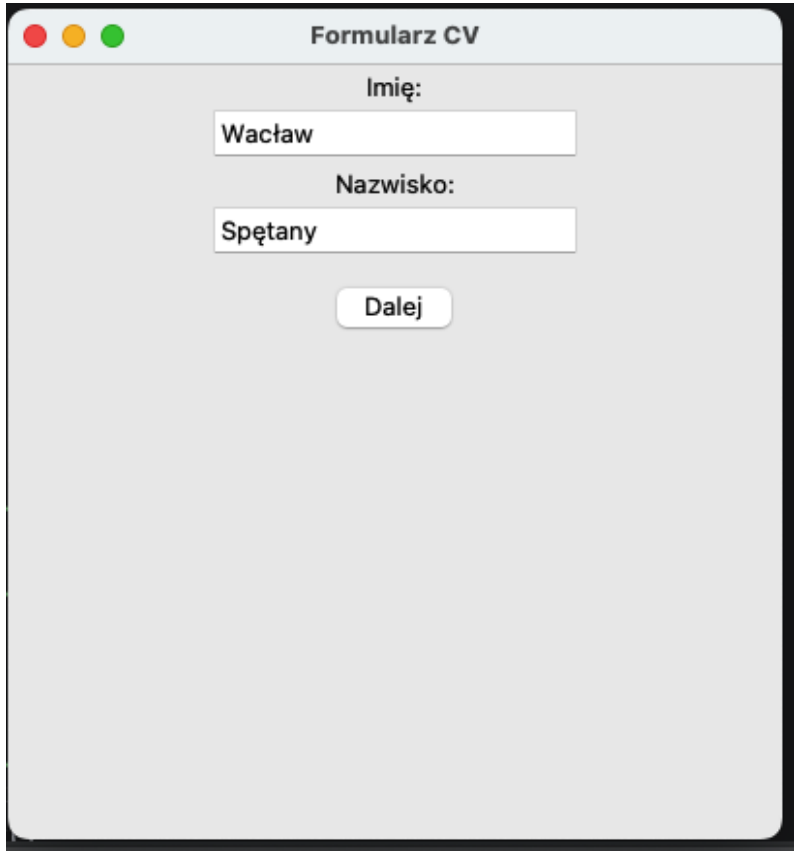


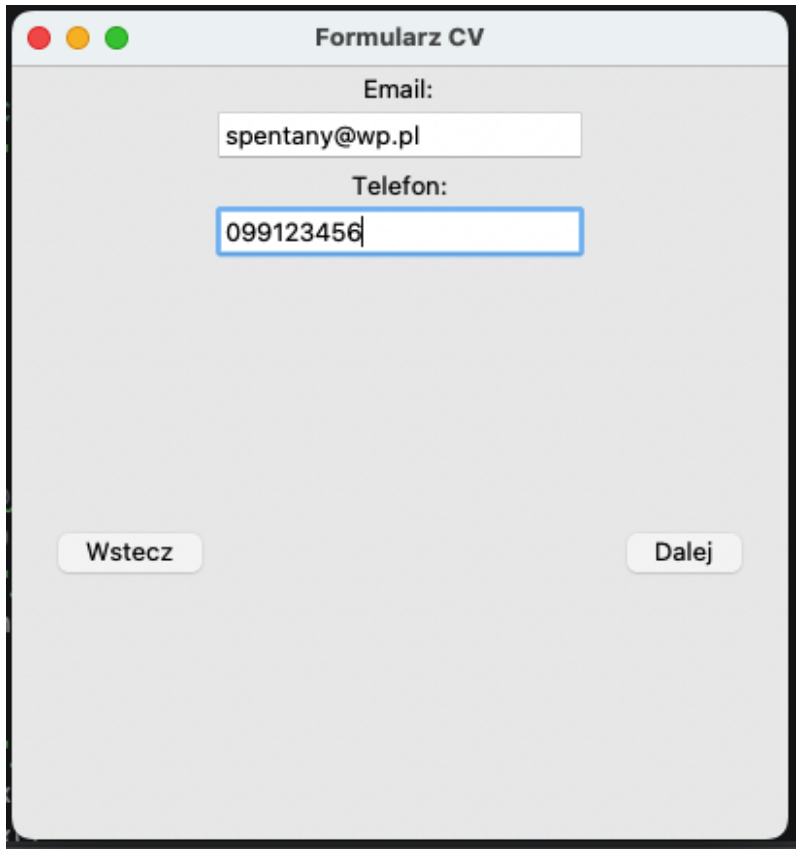
Zadania do samodzielnego wykonania



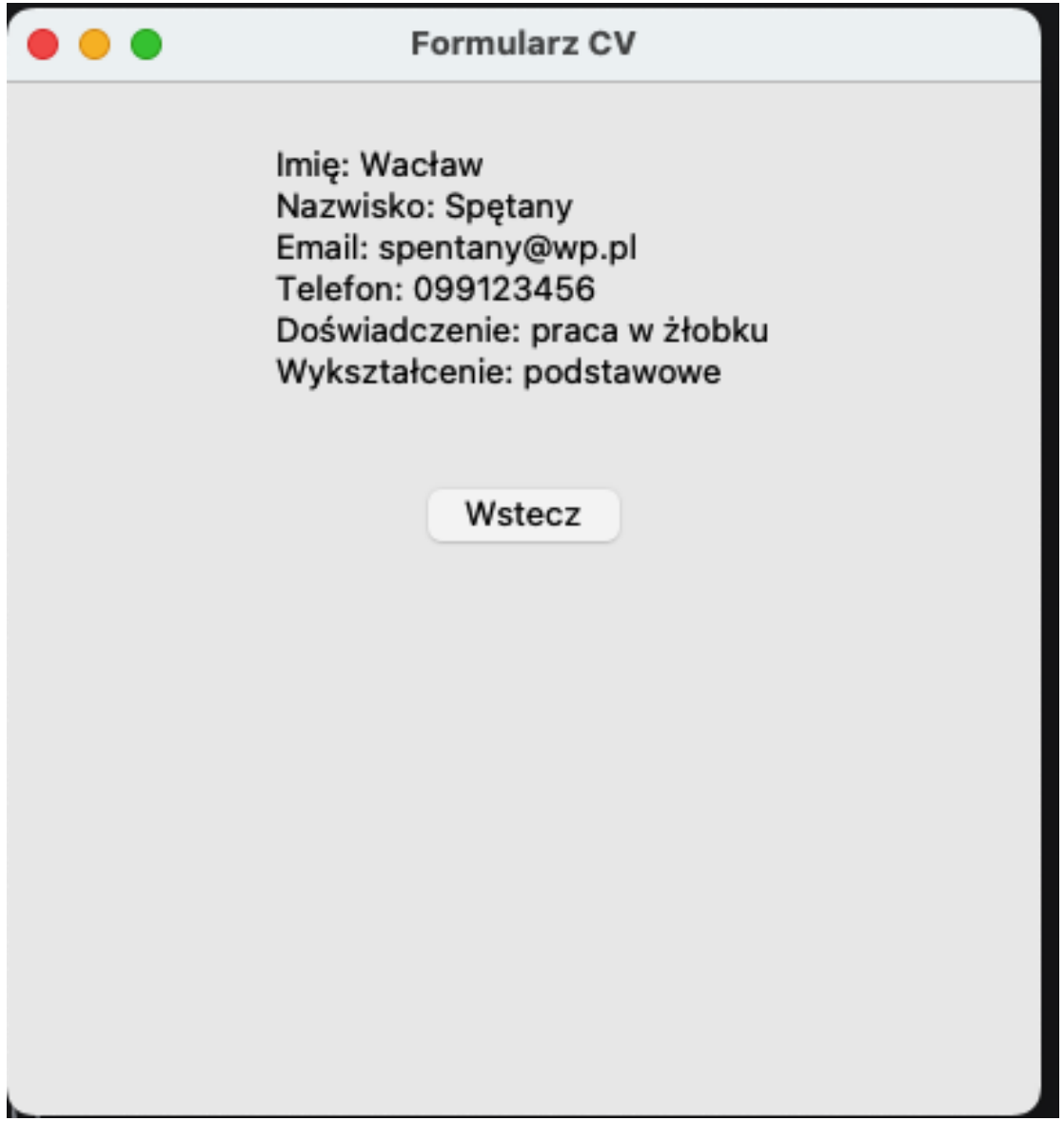
Zadanie 3

Wykonaj program w postaci listy zadań. Użytkownik wprowadza dane w postaci tekstu. Po naciśnięciu przycisku do zadania zostaje dodana data i godzina oraz zostaje zapisane w liście. Przycisk „Wyczyść” usuwa pojedynczy wpis.





Zadanie do samodzielnego wykonania na 5.



Zadanie do samodzielnego wykonania na 5.

Zadanie 4

Wykonaj aplikację w formie formularza zbierającego dane i wyświetlającego je na ostatnim oknie.

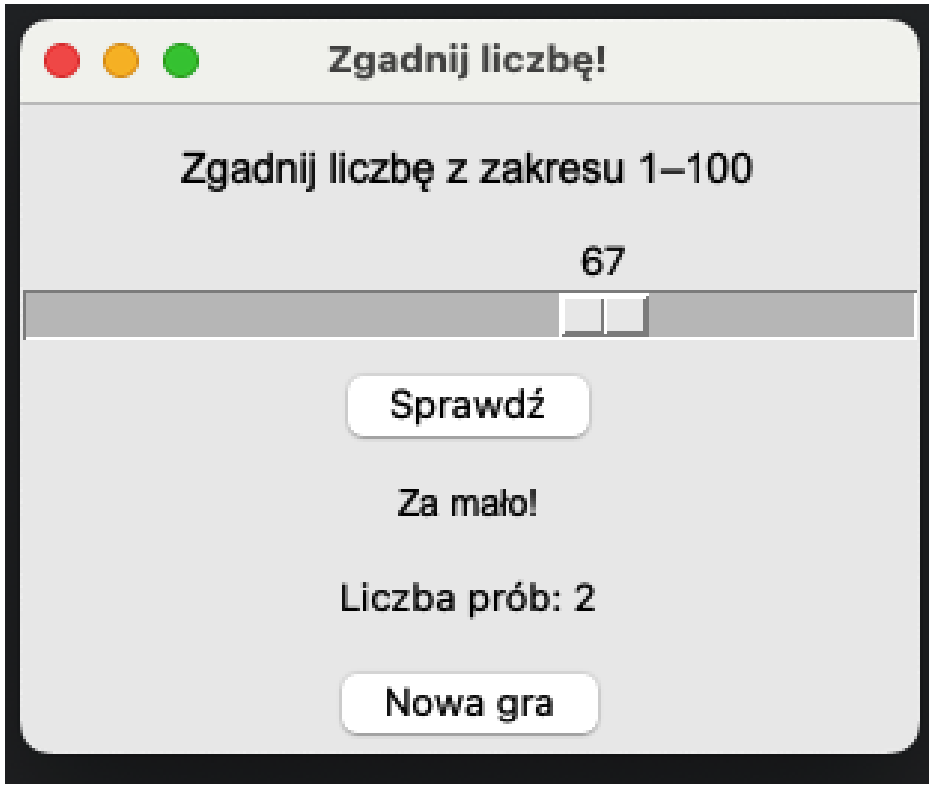
Uwaga!

W tym projekcie należy zastosować kilka funkcji, będzie konieczny również „SUPER” oraz tzw. „podwójna podłoga”, która pomoże nam zainicjować metody.

Zadania do samodzielnego wykonania

Zadanie 5

Wykonaj zadanie, w którym użytkownik za pomocą suwaka wybiera liczbę pomiędzy 1 a 100. Po naciśnięciu przycisku program sprawdza czy użytkownik podał liczbę większą od tej, którą wylosował program. Każda próba zgadywania liczby jest zliczania.



Zadania do samodzielnego wykonania

Zadanie 6

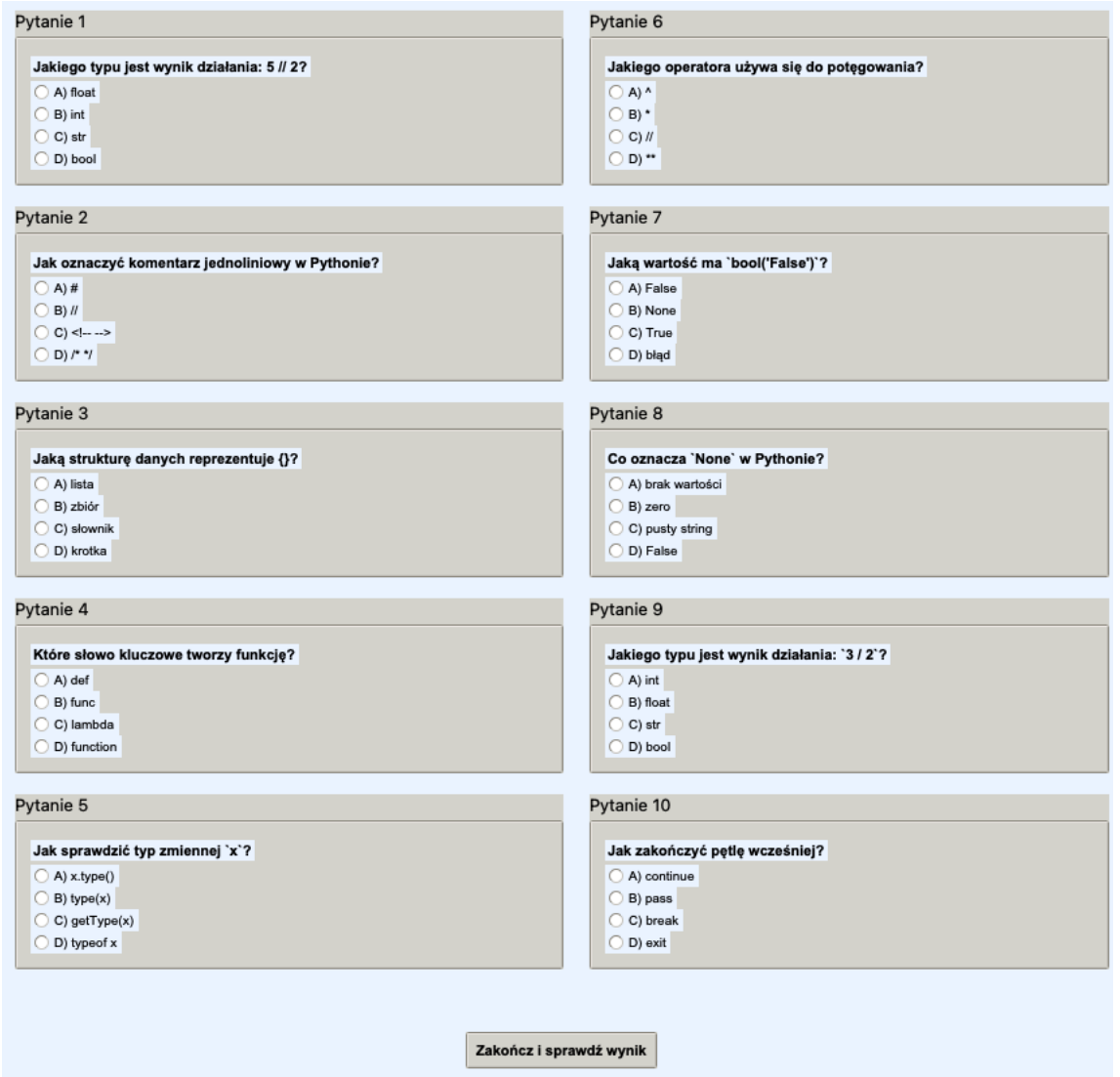
Na podstawie wzoru utwórz aplikację, której zadaniem będzie wyszukiwanie informacji o pracownikach. Użytkownik może przeszukiwać dane za pomocą prostej wyszukiwarki oraz naciskając przyciski funkcyjne „poprzedni i „następny”.

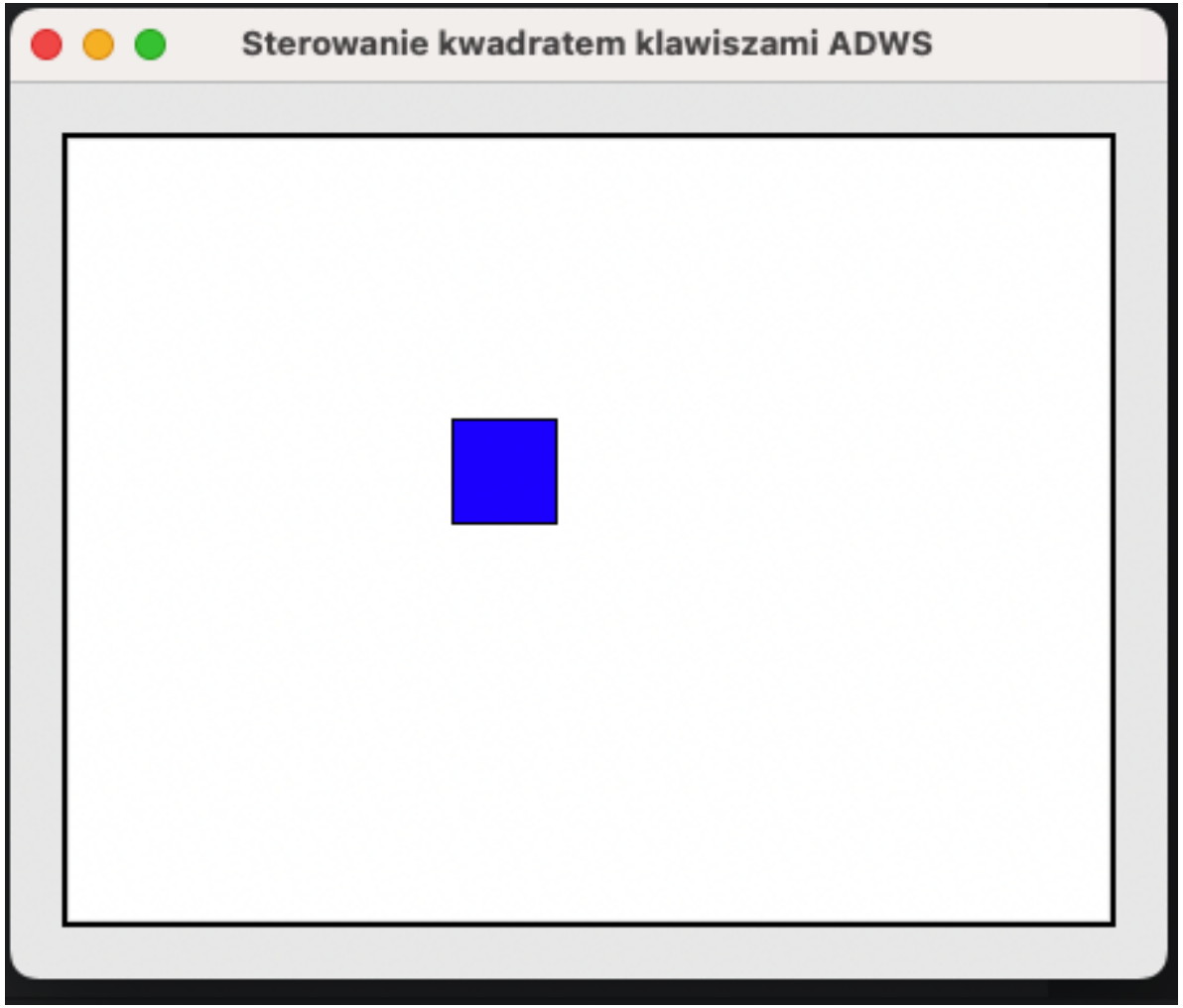
Zadania do samodzielnego wykonania

Zadanie 7

Przed Tobą test z Pythona. Nie, zrób to sam. Okno zawiera pytania w dwóch kolumnach. Jedna odpowiedź jest prawidłowa. Elementy sformatowano segregując je w grupy.

W kodzie należy umieścić punktację i odpowiadające jej oceny. Wynik wyświetla się w oddzielnym okienku.





Zadania do samodzielnego wykonania

Zadanie 8

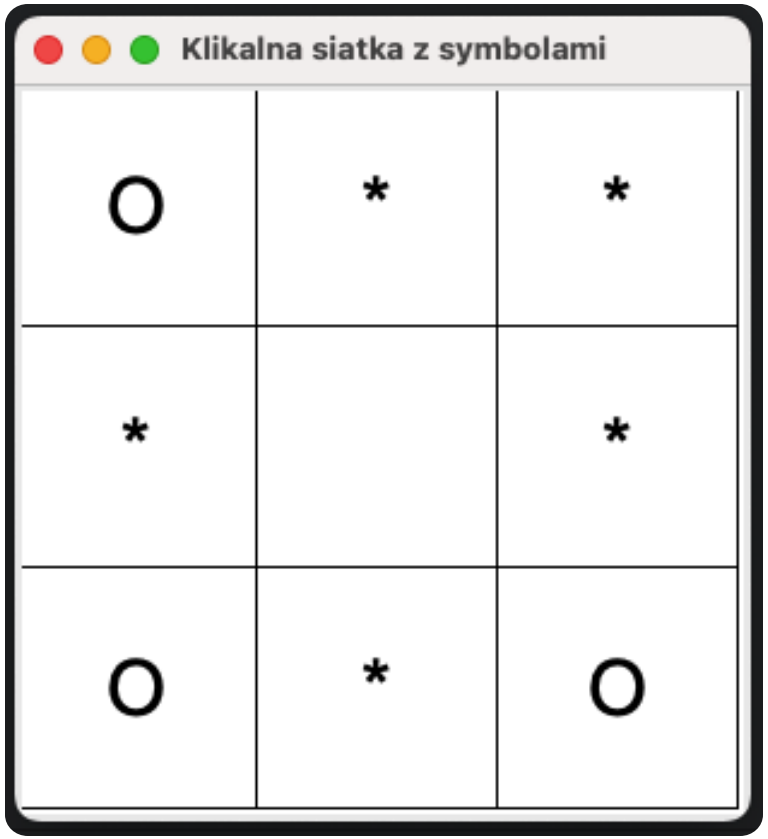
Napisz program, w którym użytkownik za pomocą klawiszy ADWS steruje kwadratem. Ruch w każdym kierunku to 10px. Wykorzystaj zdarzenia klawiszy oraz Canvas.

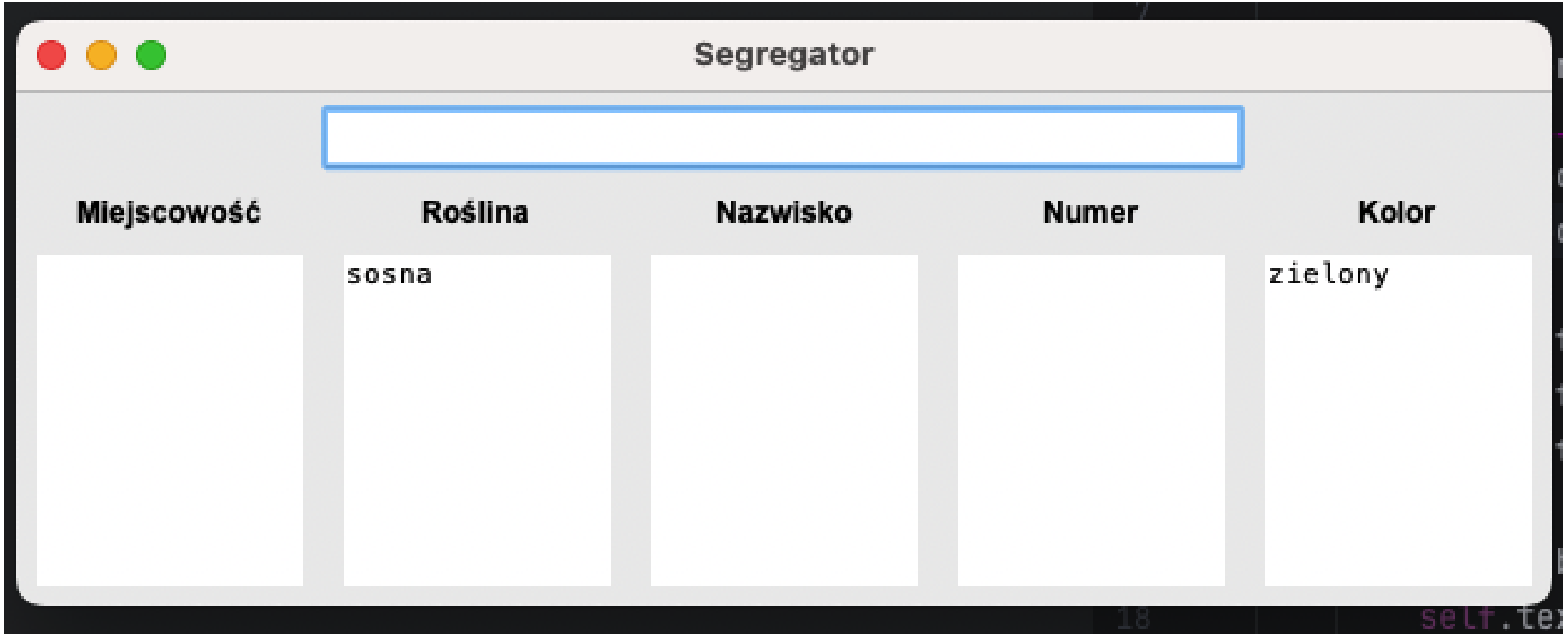
Zadania do samodzielnego wykonania

Zadanie 9

Aplikacja wykonana dzięki Canvas dzieli okno na 9 pól. Za każdym naciśnięciem symbol w polu zmienia się na:

- po pierwszym kliknięciu na z „X” na „O”;
- po drugim kliknięciu z „O” na „*”;
- po trzecim z „*” na puste pole.

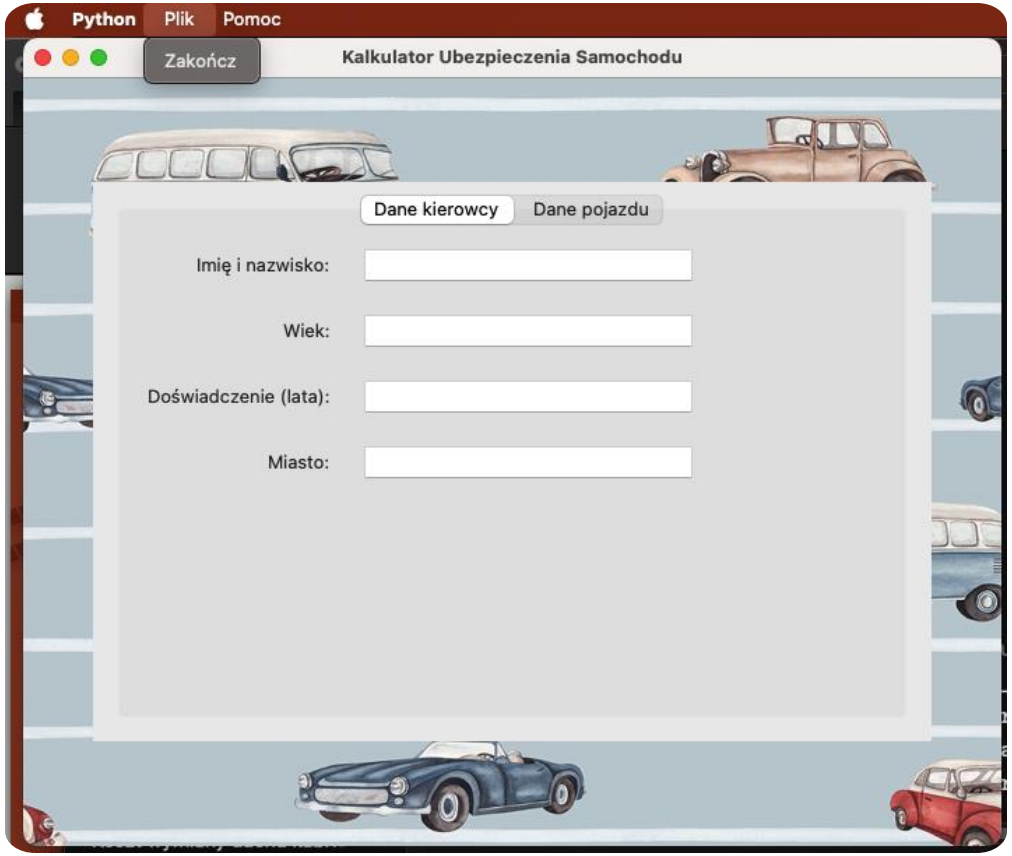




Zadania do samodzielnego wykonania

Zadanie 10

Użytkownik podaje wybrany wyraz, następnie program sprawdza czy znajduje się na liście i przydziela do właściwej kategorii.



Zadanie do samodzielnego wykonania

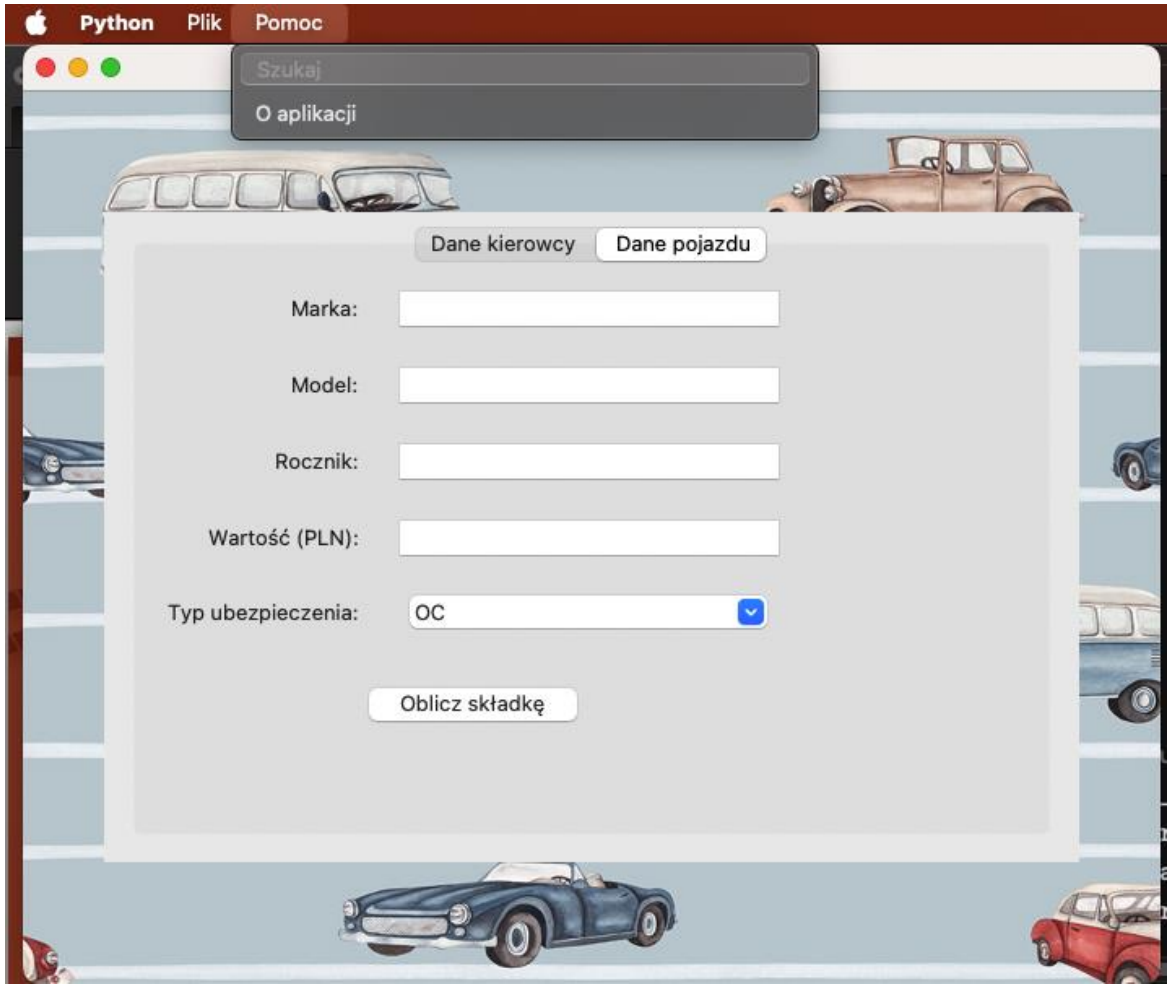
Zadanie 11 (na 5)

Aplikacja oblicza składkę polisy ubezpieczeniowej. Zawiera dwie zakładki oraz menu. Składka powinna być uzależniona od wieku oraz od wartości pojazdu. Do obliczenia całej polisy stosuje się mnożnik:

- AC to około 20% wartości polisy OC
- OC+AC to 120% wartości polisy OC
- Pełne ubezpieczenie (pakiet) to dwukrotność OC.

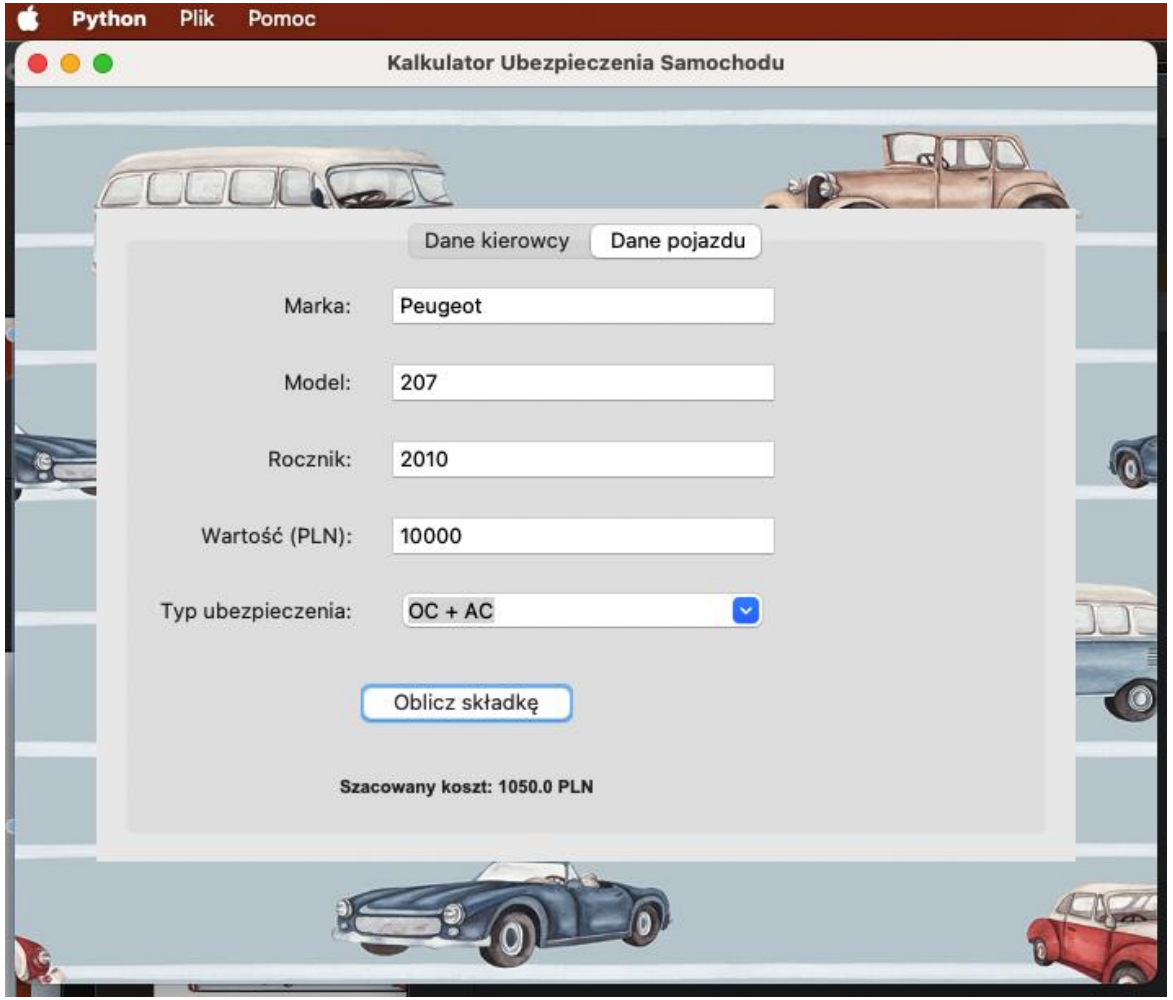
Zadanie do samodzielnego wykonania

Użytkownik wybiera rodzaj polisy. Cena wyświetla się pod przyciskiem . Menu zawiera jedynie opis programu w menu „Pomoc” oraz zakończenie działania w meny „Plik”.



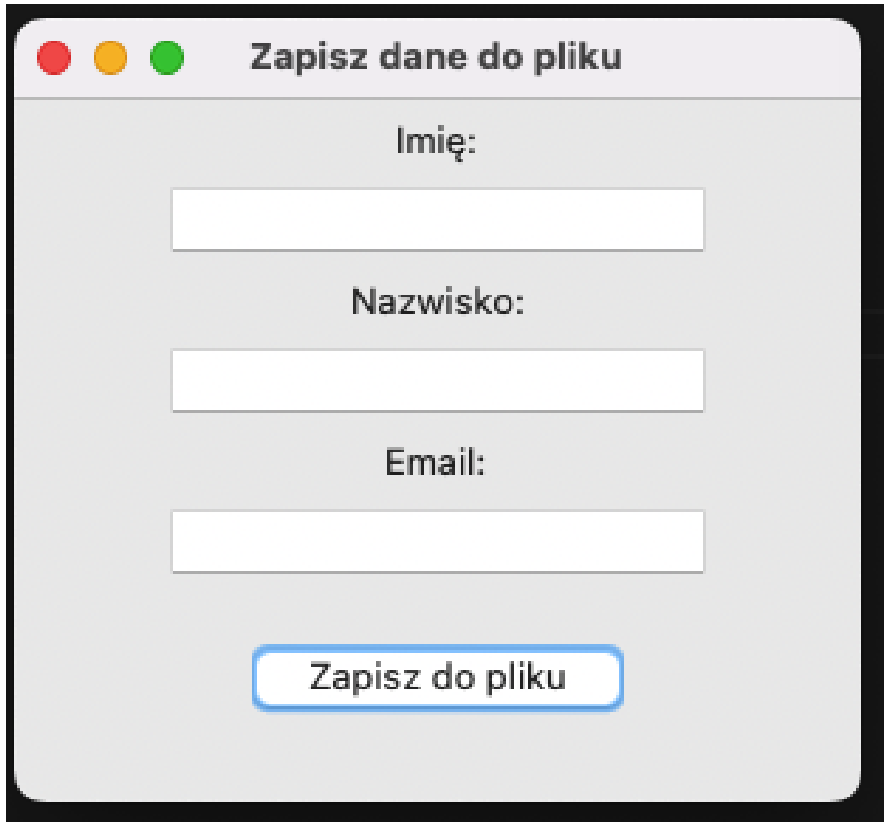
Zadanie do samodzielnego wykonania

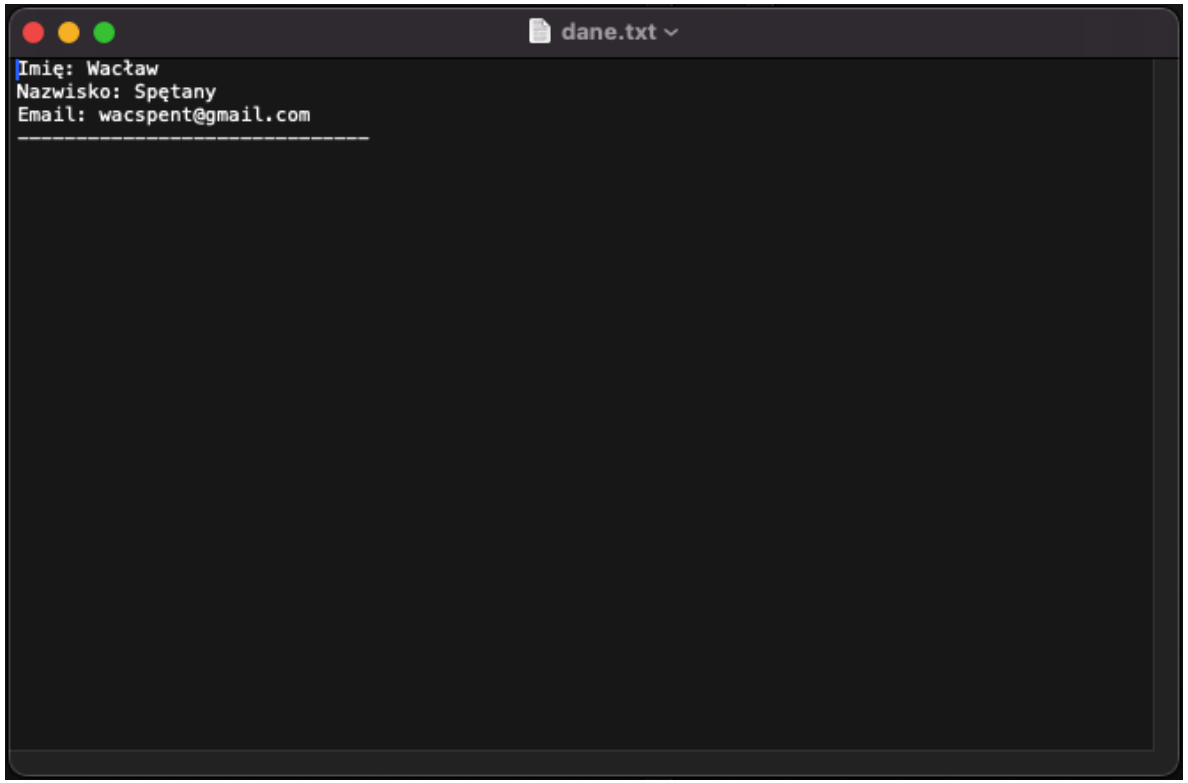
Druga karta



Zapis danych do pliku

```
import tkinter as tk
from tkinter import ttk
from tkinter import messagebox

def zapisz_dane():
dane = {
"Imię": entry_imie.get(),
"Nazwisko": entry_nazwisko.get(),
"Email": entry_email.get()
}

# Sprawdzenie, czy wszystkie pola są wypełnione
if all(dane.values()):
try:
with open("dane.txt", "a", encoding="utf-8") as plik:
for klucz, wartosc in dane.items():
plik.write(f"{klucz}: {wartosc}\n")
plik.write("-" * 30 + "\n")
messagebox.showinfo("Sukces", "Dane zostały zapisane do pliku.")
entry_imie.delete(0, tk.END)
entry_nazwisko.delete(0, tk.END)
entry_email.delete(0, tk.END)
except Exception as e:
messagebox.showerror("Błąd", f"Nie udało się zapisać danych:\n{e}")
else:
messagebox.showwarning("Uwaga", "Wypełnij wszystkie pola!")

# Główne okno
root = tk.Tk()
root.title("Zapisz dane do pliku")
root.geometry("300x250")

# Etykiety i pola tekstowe
ttk.Label(root, text="Imię:").pack(pady=5)
entry_imie = ttk.Entry(root)
entry_imie.pack()

ttk.Label(root, text="Nazwisko:").pack(pady=5)
entry_nazwisko = ttk.Entry(root)
entry_nazwisko.pack()

ttk.Label(root, text="Email:").pack(pady=5)
entry_email = ttk.Entry(root)
entry_email.pack()

# Przycisk zapisu
ttk.Button(root, text="Zapisz do pliku", command=zapisz_dane).pack(pady=20)

root.mainloop()
```

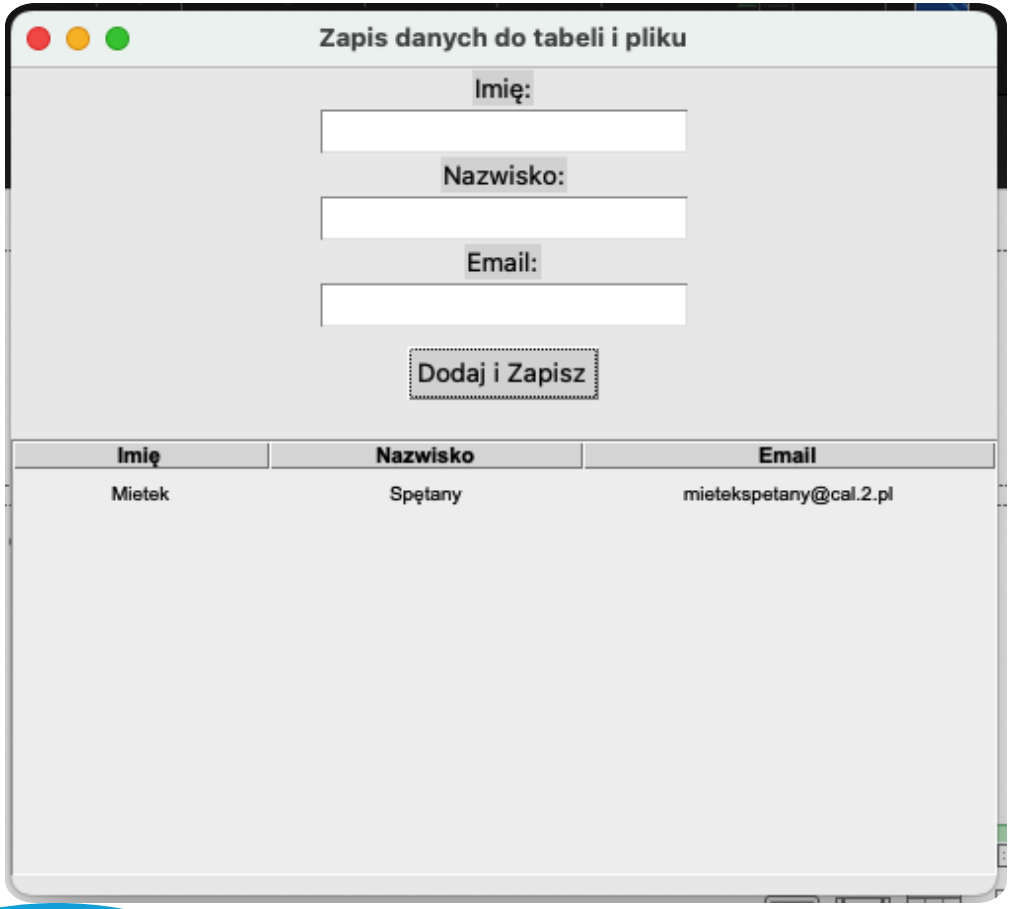

Zapisz dane do pliku
Imię:
Nazwisko:
Email:
Zapisz do pliku


dane.txt
Imię: Wacław
Nazwisko: Spętany
Email: wacspent@gmail.com

Efekt?

Zapis uporządkowany - tabela

```
import tkinter as tk
from tkinter import ttk, messagebox
import csv
import os

def dodaj_dane():
imie = entry_imie.get()
nazwisko = entry_nazwisko.get()
email = entry_email.get()

if imie and nazwisko and email:
tree.insert("", "end", values=(imie, nazwisko, email))

with open("dane.csv", "a", newline="", encoding="utf-8") as plik:
writer = csv.writer(plik)
writer.writerow([imie, nazwisko, email])

entry_imie.delete(0, tk.END)
entry_nazwisko.delete(0, tk.END)
entry_email.delete(0, tk.END)
else:
messagebox.showwarning("Uwaga", "Wypełnij wszystkie pola!")

# Okno główne
root = tk.Tk()
root.title("Zapis danych do tabeli i pliku")
root.geometry("500x420")

# Stylizacja Treeview
style = ttk.Style()
style.theme_use("default")
style.configure("Treeview",
background="#f0f0f0",
foreground="black",
rowheight=25,
fieldbackground="#f0f0f0",
font=("Arial", 10)
)
style.configure("Treeview.Heading",
font=("Arial", 11, "bold"),
background="#dcdcdc"
)
style.map("Treeview",
background=[("selected", "#007acc")]
)

# Naprzemienne kolory wierszy
style.map("Treeview",
background=[("selected", "#007acc")])
tree_tag_even = "evenrow"
tree_tag_odd = "oddrow"

# Pola do wpisania danych
ttk.Label(root, text="Imię:").pack(pady=2)
entry_imie = ttk.Entry(root)
entry_imie.pack()

ttk.Label(root, text="Nazwisko:").pack(pady=2)
entry_nazwisko = ttk.Entry(root)
entry_nazwisko.pack()

ttk.Label(root, text="Email:").pack(pady=2)
entry_email = ttk.Entry(root)
entry_email.pack()

ttk.Button(root, text="Dodaj i Zapisz", command=dodaj_dane).pack(pady=10)

# Tabela (Treeview)
tree = ttk.Treeview(root, columns=("Imię", "Nazwisko", "Email"), show="headings", selectmode="browse")
tree.heading("Imię", text="Imię")
tree.heading("Nazwisko", text="Nazwisko")
tree.heading("Email", text="Email")

# Szerokości kolumn
tree.column("Imię", width=120, anchor="center")
tree.column("Nazwisko", width=150, anchor="center")
tree.column("Email", width=200, anchor="center")

tree.pack(expand=True, fill="both", pady=10)

# Dodaj nagłówki do pliku CSV jeśli nie istnieje
if not os.path.exists("dane.csv"):
with open("dane.csv", "w", newline="", encoding="utf-8") as plik:
writer = csv.writer(plik)
writer.writerow(["Imię", "Nazwisko", "Email"])

root.mainloop()
```



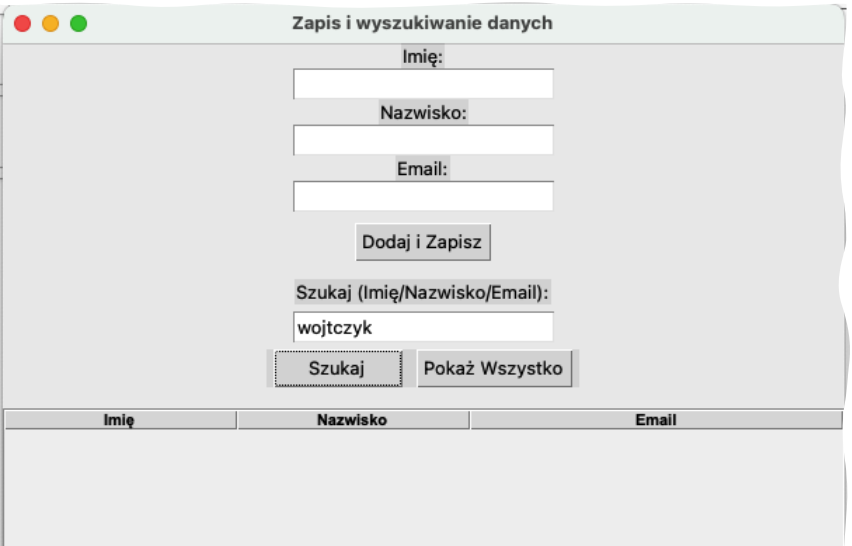
Efekt?

Program jednocześnie zapisuje dane do tabeli i pliku z rozszerzeniem CSV. Za pomocą arkusza kalkulacyjnego można wyświetlić dane w estetyczny sposób.

A jak wyszukać dane z pliku?

Wyszukiwarka

```
import tkinter as tk
from tkinter import ttk, messagebox
import csv
import os

plik_csv = "dane.csv"

def dodaj_dane():
imie = entry_imie.get()
nazwisko = entry_nazwisko.get()
email = entry_email.get()

if imie and nazwisko and email:
# Dodaj do tabeli
tree.insert("", "end", values=(imie, nazwisko, email))

# Zapisz do pliku
with open(plik_csv, "a", newline="", encoding="utf-8") as plik:
writer = csv.writer(plik)
writer.writerow([imie, nazwisko, email])

# Wyczyść pola
entry_imie.delete(0, tk.END)
entry_nazwisko.delete(0, tk.END)
entry_email.delete(0, tk.END)
else:
messagebox.showwarning("Uwaga", "Wypełnij wszystkie pola!")

def wczytaj_dane():
# Wczytaj wszystkie dane z pliku
tree.delete(*tree.get_children())
if os.path.exists(plik_csv):
with open(plik_csv, newline="", encoding="utf-8") as plik:
reader = csv.reader(plik)
next(reader, None) # Pomijamy nagłówek
for row in reader:
if len(row) == 3:
tree.insert("", "end", values=row)

def szukaj_danych():
query = entry_szukaj.get().lower()
tree.delete(*tree.get_children())
if os.path.exists(plik_csv):
with open(plik_csv, newline="", encoding="utf-8") as plik:
reader = csv.reader(plik)
next(reader, None) # Pomijamy nagłówek
for row in reader:
if any(query in field.lower() for field in row):
tree.insert("", "end", values=row)

# === GUI ===

root = tk.Tk()
root.title("Zapis i wyszukiwanie danych")
root.geometry("600x500")

# Stylizacja
style = ttk.Style()
style.theme_use("default")
style.configure("Treeview", background="#f0f0f0", foreground="black", rowheight=25, fieldbackground="#f0f0f0", font=("Arial", 10))
style.configure("Treeview.Heading", font=("Arial", 11, "bold"), background="#dcdcdc")
style.map("Treeview", background=[("selected", "#007acc")])

# Pola danych
ttk.Label(root, text="Imię:").pack()
entry_imie = ttk.Entry(root)
entry_imie.pack()

ttk.Label(root, text="Nazwisko:").pack()
entry_nazwisko = ttk.Entry(root)
entry_nazwisko.pack()

ttk.Label(root, text="Email:").pack()
entry_email = ttk.Entry(root)
entry_email.pack()

ttk.Button(root, text="Dodaj i Zapisz", command=dodaj_dane).pack(pady=8)

# Wyszukiwarka
ttk.Label(root, text="Szukaj (Imię/Nazwisko/Email):").pack(pady=5)
entry_szukaj = ttk.Entry(root)
entry_szukaj.pack()

frame_btns = ttk.Frame(root)
frame_btns.pack(pady=5)

ttk.Button(frame_btns, text="Szukaj", command=szukaj_danych).pack(side="left", padx=5)
ttk.Button(frame_btns, text="Pokaż Wszystko", command=wczytaj_dane).pack(side="left", padx=5)

# Tabela
tree = ttk.Treeview(root, columns=("Imię", "Nazwisko", "Email"), show="headings", selectmode="browse")
tree.heading("Imię", text="Imię")
tree.heading("Nazwisko", text="Nazwisko")
tree.heading("Email", text="Email")
tree.column("Imię", width=150, anchor="center")
tree.column("Nazwisko", width=150, anchor="center")
tree.column("Email", width=250, anchor="center")
tree.pack(expand=True, fill="both", pady=10)

# Tworzenie pliku z nagłówkiem (jeśli nie istnieje)
if not os.path.exists(plik_csv):
with open(plik_csv, "w", newline="", encoding="utf-8") as plik:
writer = csv.writer(plik)
writer.writerow(["Imię", "Nazwisko", "Email"])

wczytaj_dane()
root.mainloop()
```



Efekt?

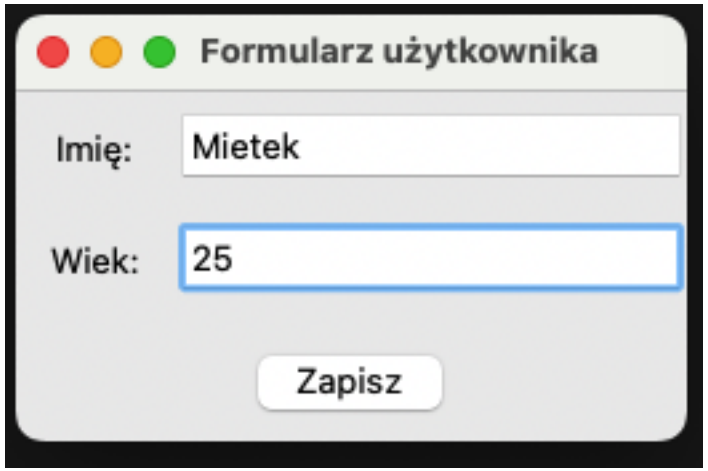
Rozszerzenie funkcjonalności o wyszukiwarkę, która wyszukuje po każdym wprowadzonym znaku i w tabeli wyświetla proponowane rekordy.

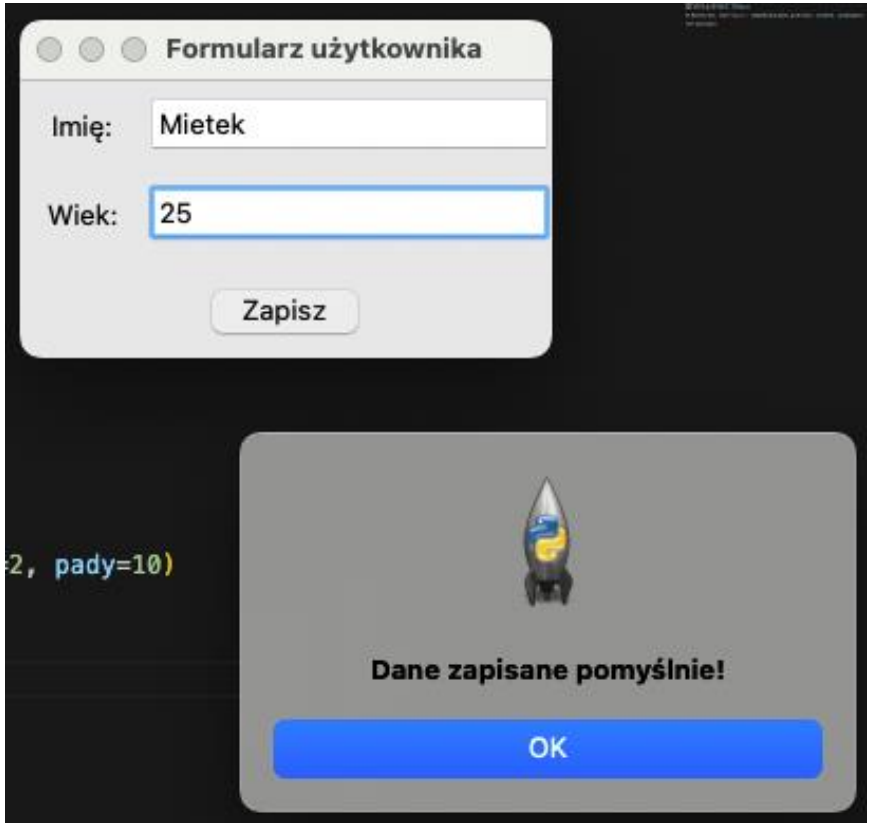
Tabela jest pusta, gdyż nie ma rekordu spełniającego kryteria.

Zapis danych do bazy (SQL)

```
import tkinter as tk
from tkinter import messagebox
import mysql.connector

# Dane połączenia do MySQL w MAMP na MacOS
DB_CONFIG = {
'host': 'localhost',
'port': 8889,
'user': 'root',
'password': 'root',
'database': 'python' # Upewnij się, że ta baza istnieje w
phpMyAdmin
}

# Tworzenie tabeli
def init_db():
try:
conn = mysql.connector.connect(**DB_CONFIG)
cursor = conn.cursor()
cursor.execute('''
CREATE TABLE IF NOT EXISTS users (
id INT AUTO_INCREMENT PRIMARY KEY,
name VARCHAR(255) NOT NULL,
age INT NOT NULL
)
''')
conn.commit()
conn.close()
except mysql.connector.Error as err:
messagebox.showerror("Błąd bazy danych", f"Błąd
połączenia:\n{err}")

# Zapis danych
def save_data():
name = name_entry.get()
age = age_entry.get()

if not name or not age:
messagebox.showwarning("Błąd", "Wypełnij wszystkie pola!")
return

try:
age = int(age)
except ValueError:
messagebox.showerror("Błąd", "Wiek musi być liczbą!")
return

try:
conn = mysql.connector.connect(**DB_CONFIG)
cursor = conn.cursor()
cursor.execute("INSERT INTO users (name, age) VALUES (%s,
%s)", (name, age))
conn.commit()
conn.close()
messagebox.showinfo("Sukces", "Dane zapisane pomyślnie!")
name_entry.delete(0, tk.END)
age_entry.delete(0, tk.END)
except mysql.connector.Error as err:
messagebox.showerror("Błąd zapisu", f"{err}")

# Inicjalizacja bazy
init_db()

# GUI
root = tk.Tk()
root.title("Formularz użytkownika")

tk.Label(root, text="Imię:").grid(row=0, column=0, padx=10,
pady=10)
name_entry = tk.Entry(root)
name_entry.grid(row=0, column=1)

tk.Label(root, text="Wiek:").grid(row=1, column=0, padx=10,
pady=10)
age_entry = tk.Entry(root)
age_entry.grid(row=1, column=1)

tk.Button(root, text="Zapisz", command=save_data).grid(row=2,
column=0, columnspan=2, pady=10)

root.mainloop()
```

Efekt działania kodu ...





Wersja z wyświetleniem w postaci tabeli

```
import tkinter as tk
from tkinter import ttk, messagebox
import mysql.connector

#  Konfiguracja bazy MySQL z MAMP
DB_CONFIG = {
'host': 'localhost',
'port': 8889,
'user': 'root',
'password': 'root',
'database': 'python' # upewnij się, że baza istnieje
}

# Inicjalizacja bazy danych i tabeli
def init_db():
try:
conn = mysql.connector.connect(**DB_CONFIG)
cursor = conn.cursor()
cursor.execute("'
CREATE TABLE IF NOT EXISTS users (
id INT AUTO_INCREMENT PRIMARY KEY,
name VARCHAR(255) NOT NULL,
age INT NOT NULL
)
'")
conn.commit()
conn.close()
except mysql.connector.Error as err:
messagebox.showerror("Błąd bazy danych", f"Nie
można utworzyć tabeli:\n{err}")

#  Zapis danych do bazy
def save_data():
name = name_entry.get()
age = age_entry.get()

if not name or not age:
messagebox.showwarning("Błąd", "Wypełnij wszystkie
pola!")
return

try:
age = int(age)
except ValueError:
messagebox.showerror("Błąd", "Wiek musi być liczbą!")
return

try:
conn = mysql.connector.connect(**DB_CONFIG)
cursor = conn.cursor()
cursor.execute("INSERT INTO users (name, age)
VALUES (%s, %s)", (name, age))
conn.commit()
conn.close()
messagebox.showinfo("Sukces", "Dane zapisane
pomyślnie!")
name_entry.delete(0, tk.END)
age_entry.delete(0, tk.END)
load_data() # odśwież dane po dodaniu
except mysql.connector.Error as err:
messagebox.showerror("Błąd zapisu", f"{err}")

#  Pobieranie i wyświetlanie danych
def load_data():
for row in tree.get_children():
tree.delete(row)

try:
conn = mysql.connector.connect(**DB_CONFIG)
cursor = conn.cursor()
cursor.execute("SELECT id, name, age FROM users")
rows = cursor.fetchall()
conn.close()

for row in rows:
tree.insert('', 'end', values=row)
except mysql.connector.Error as err:
messagebox.showerror("Błąd odczytu", f"{err}")

#  Budowa GUI
root = tk.Tk()
root.title("Rejestracja i lista użytkowników")

# Sekcja: Formularz
form_frame = tk.Frame(root)
form_frame.pack(padx=10, pady=10)

tk.Label(form_frame, text="Imię:").grid(row=0,
column=0, padx=5, pady=5)
name_entry = tk.Entry(form_frame)
name_entry.grid(row=0, column=1, padx=5, pady=5)

tk.Label(form_frame, text="Wiek:").grid(row=1,
column=0, padx=5, pady=5)
age_entry = tk.Entry(form_frame)
age_entry.grid(row=1, column=1, padx=5, pady=5)

save_button = tk.Button(form_frame, text="Zapisz",
command=save_data)
save_button.grid(row=2, column=0, columnspan=2,
pady=10)

# Sekcja: Tabela z użytkownikami
tree_frame = tk.Frame(root)
tree_frame.pack(padx=10, pady=10, fill=tk.BOTH,
expand=True)

refresh_button = tk.Button(tree_frame, text="Odśwież
dane", command=load_data)
refresh_button.pack(pady=5)

tree = ttk.Treeview(tree_frame, columns=('ID', 'Imię',
'Wiek'), show='headings')
tree.heading('ID', text='ID')
tree.heading('Imię', text='Imię')
tree.heading('Wiek', text='Wiek')
tree.pack(fill=tk.BOTH, expand=True)

# Uruchomienie
init_db()
load_data()
root.mainloop()
```

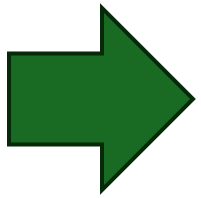
Efekt działania kodu...

Użytkownik wprowadza dane. Następnie zostają zapisane w bazie. Po naciśnięciu przycisku wprowadzone dane zostają wyświetlone w tabeli.

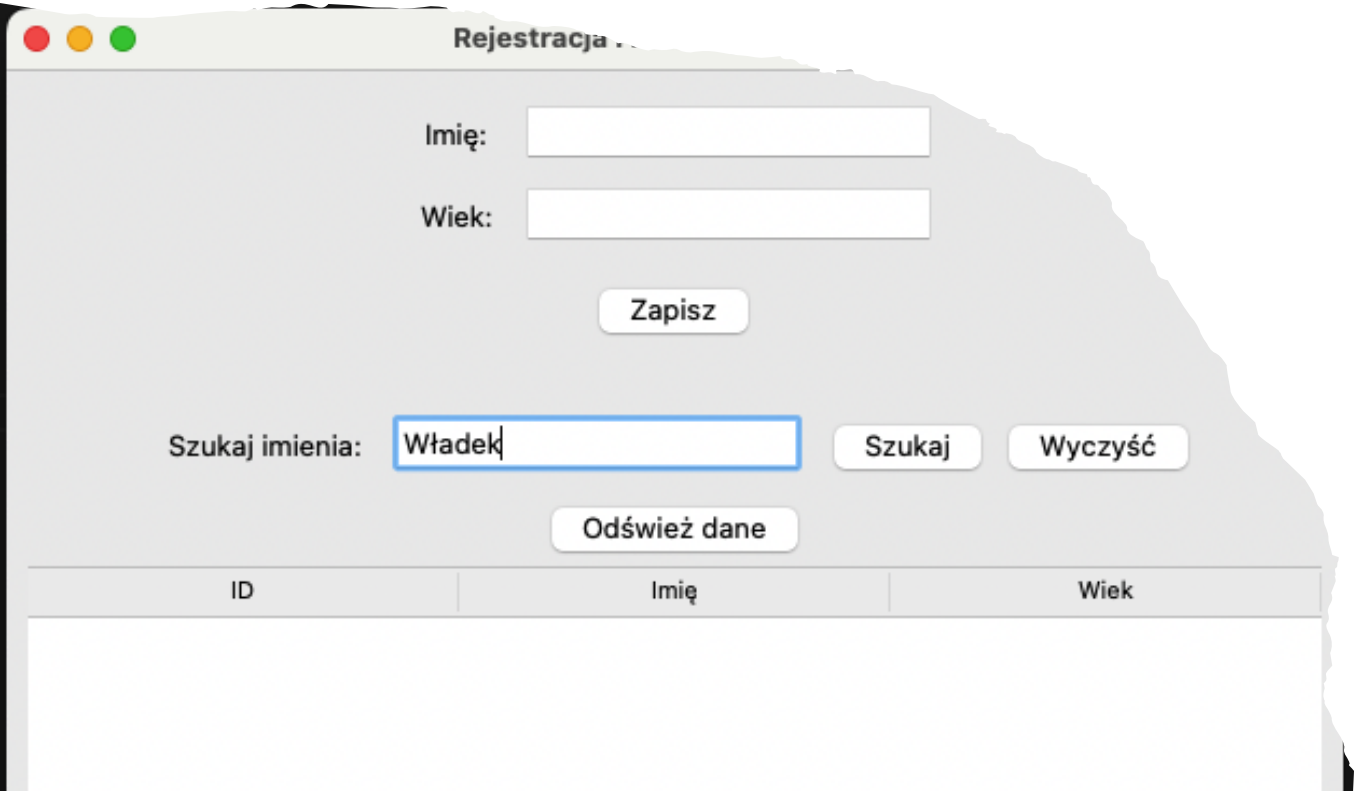
Uwaga:
Przykład oparto o serwer MAMP i środowisko. MacOS. W Windows domyślnym portem jest 3306.
W Windows nie jest konieczne domyślne hasło użytkownika. Może być puste.
W MacOS jest to wymóg bezpieczeństwa.

Dodanie wyszukiwarki

```
import tkinter as tk
from tkinter import ttk, messagebox
import mysql.connector

DB_CONFIG = {
'host': 'localhost',
'port': 8889,
'user': 'root',
'password': 'root',
'database': 'python'
}

def init_db():
try:
conn = mysql.connector.connect(**DB_CONFIG)
cursor = conn.cursor()
cursor.execute("'
CREATE TABLE IF NOT EXISTS users (
id INT AUTO_INCREMENT PRIMARY KEY,
name VARCHAR(255) NOT NULL,
age INT NOT NULL
)
'")
conn.commit()
conn.close()
except mysql.connector.Error as err:
messagebox.showerror("Błąd bazy danych", f"Nie można utworzyć tabeli:\n{err}")

def save_data():
name = name_entry.get()
age = age_entry.get()

if not name or not age:
messagebox.showwarning("Błąd", "Wypełnij wszystkie pola!")
return

try:
age = int(age)
except ValueError:
messagebox.showerror("Błąd", "Wiek musi być liczbą!")
return

try:
conn = mysql.connector.connect(**DB_CONFIG)
cursor = conn.cursor()
cursor.execute("INSERT INTO users (name, age) VALUES (%s, %s)", (name, age))
conn.commit()
conn.close()
messagebox.showinfo("Sukces", "Dane zapisane pomyślnie!")
name_entry.delete(0, tk.END)
age_entry.delete(0, tk.END)
load_data() # odśwież dane po dodaniu
except mysql.connector.Error as err:
messagebox.showerror("Błąd zapisu", f"{err}")

def load_data(filter_text=None):
for row in tree.get_children():
tree.delete(row)

try:
conn = mysql.connector.connect(**DB_CONFIG)
cursor = conn.cursor()
if filter_text:
cursor.execute("SELECT id, name, age FROM users WHERE name LIKE %s", (f'%{filter_text}%',))
else:
cursor.execute("SELECT id, name, age FROM users")
rows = cursor.fetchall()
conn.close()

for row in rows:
tree.insert('', 'end', values=row)
except mysql.connector.Error as err:
messagebox.showerror("Błąd odczytu", f"{err}")

def search_data():
query = search_entry.get().strip()
load_data(filter_text=query)

def clear_search():
search_entry.delete(0, tk.END)
load_data()

root = tk.Tk()
root.title("Rejestracja i lista użytkowników")

form_frame = tk.Frame(root)
form_frame.pack(padx=10, pady=10)

tk.Label(form_frame, text="Imię:").grid(row=0, column=0, padx=5, pady=5)
name_entry = tk.Entry(form_frame)
name_entry.grid(row=0, column=1, padx=5, pady=5)

tk.Label(form_frame, text="Wiek:").grid(row=1, column=0, padx=5, pady=5)
age_entry = tk.Entry(form_frame)
age_entry.grid(row=1, column=1, padx=5, pady=5)

save_button = tk.Button(form_frame, text="Zapisz", command=save_data)
save_button.grid(row=2, column=0, columnspan=2, pady=10)
```



Dodanie wyszukiwarki




```
tree_frame = tk.Frame(root)
tree_frame.pack(padx=10, pady=10, fill=tk.BOTH, expand=True)

search_frame = tk.Frame(tree_frame)
search_frame.pack(pady=5)

tk.Label(search_frame, text="Szukaj imienia:").pack(side=tk.LEFT,
padx=5)
search_entry = tk.Entry(search_frame)
search_entry.pack(side=tk.LEFT, padx=5)

tk.Button(search_frame, text="Szukaj",
command=search_data).pack(side=tk.LEFT, padx=5)
tk.Button(search_frame, text="Wyczyść",
command=clear_search).pack(side=tk.LEFT)

refresh_button = tk.Button(tree_frame, text="Odśwież dane",
command=load_data)
refresh_button.pack(pady=5)

tree = ttk.Treeview(tree_frame, columns=('ID', 'Imię', 'Wiek'),
show='headings')
tree.heading('ID', text='ID')
tree.heading('Imię', text='Imię')
tree.heading('Wiek', text='Wiek')
tree.pack(fill=tk.BOTH, expand=True)

init_db()
load_data()

root.mainloop()
```

Efekt?

Całość programu okienkowego w postaci formularza, który gromadzi dane, wyświetla je i pozwala wyszukiwać. Program można rozbudować o kolejne dane. Z każdym kolejnym typem danych należy zmodyfikować tabelę w bazie SQL oraz rozbudować zapytanie.

SQLite – popularny sposób na gromadzenie danych

Python ma wbudowany moduł sqlite3, który umożliwia łatwą pracę z bazami SQLite.

Możemy tworzyć bazy, tabele, wstawiać, aktualizować i pobierać dane za pomocą standardowych zapytań SQL.

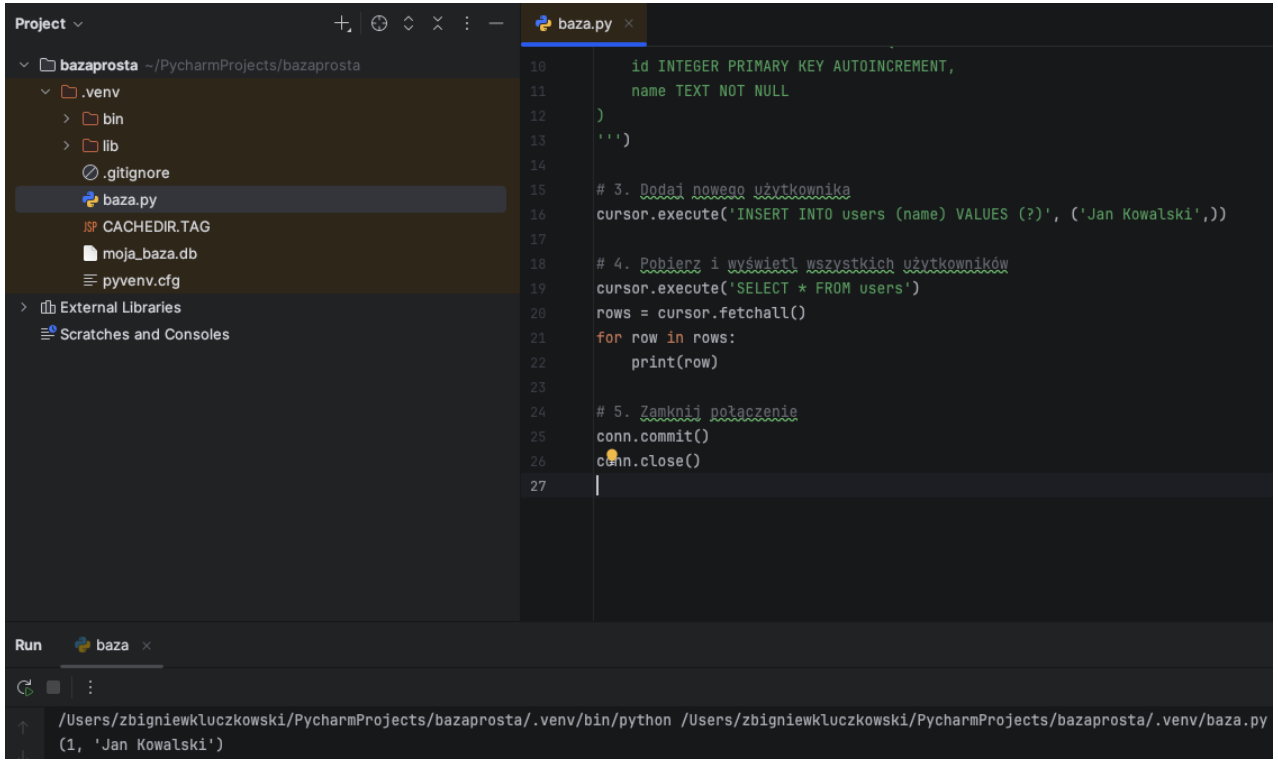
```
import sqlite3

# 1. Połącz się z bazą danych (lub utwórz, jeśli nie istnieje)
conn = sqlite3.connect('moja_baza.db')

# 2. Utwórz kursor i tabelę, jeśli nie istnieje
cursor = conn.cursor()
cursor.execute('''
CREATE TABLE IF NOT EXISTS users (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL
)
''')

# 3. Dodaj nowego użytkownika
cursor.execute('INSERT INTO users (name) VALUES (?)', ('Jan Kowalski',))

# 4. Pobierz i wyświetl wszystkich użytkowników
cursor.execute('SELECT * FROM users')
rows = cursor.fetchall()
for row in rows:
    print(row)

# 5. Zamknij połączenie
conn.commit()
conn.close()
```

Efekt?

Jak widać jest to baza umieszczona wewnątrz projektu a nie SQL, który znajduje się na Localhost.

SQLite jest prosty w obsłudze i wykorzystuje zapytania znane z „typowych” baz danych.

Jest szczególnie polecany przy przenoszeniu całej aplikacji. Mniej rozproszonych plików to tylko jedna z zalet.

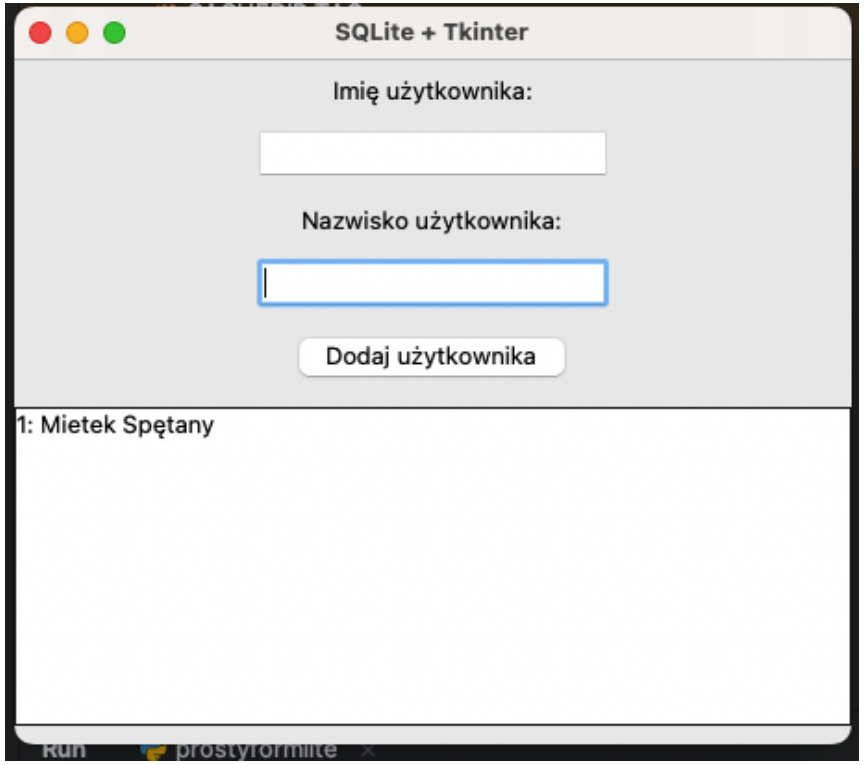


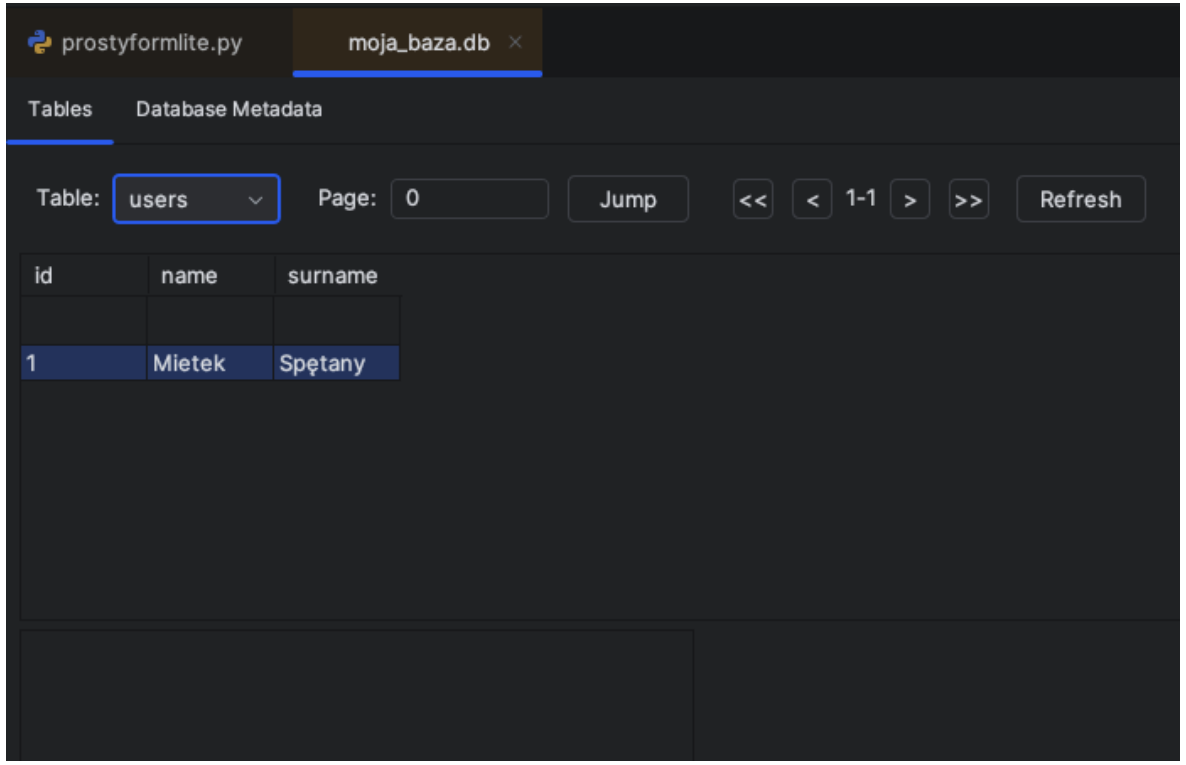
SQL i Tkinter

```
import tkinter as tk
from tkinter import messagebox
import sqlite3


DB_PATH = "moja_baza.db"


def init_db():
    conn = sqlite3.connect(DB_PATH)
    c = conn.cursor()
    c.execute('''
    CREATE TABLE IF NOT EXISTS users (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        name TEXT NOT NULL,
        surname TEXT NOT NULL
    )
    ''')
    conn.commit()
    conn.close()


def add_user():
    name = entry_name.get().strip()
    surname = entry_surname.get().strip()
    if not name or not surname:
        messagebox.showwarning("Błąd", "Podaj imię i nazwisko użytkownika!")
        return
    conn = sqlite3.connect(DB_PATH)
    c = conn.cursor()
    c.execute("INSERT INTO users (name, surname) VALUES (?, ?)", (name, surname))
    conn.commit()
    conn.close()
    entry_name.delete(0, tk.END)
    entry_surname.delete(0, tk.END)
    refresh_users()


def refresh_users():
    listbox.delete(0, tk.END)
    conn = sqlite3.connect(DB_PATH)
    c = conn.cursor()
    c.execute("SELECT id, name, surname FROM users")
    for row in c.fetchall():
        listbox.insert(tk.END, f"{row[0]}: {row[1]} {row[2]}")
    conn.close()


# --- GUI ---
root = tk.Tk()
root.title("SQLite + Tkinter")

tk.Label(root, text="Imię użytkownika:").pack(pady=5)
entry_name = tk.Entry(root)
entry_name.pack(pady=5)

tk.Label(root, text="Nazwisko użytkownika:").pack(pady=5)
entry_surname = tk.Entry(root)
entry_surname.pack(pady=5)

btn_add = tk.Button(root, text="Dodaj użytkownika", command=add_user)
btn_add.pack(pady=5)

listbox = tk.Listbox(root, width=50)
listbox.pack(pady=10)

init_db()
refresh_users()

root.mainloop()
```

Efekt działania programu

Dane są zapisywane w polu pod przyciskiem oraz w bazie.



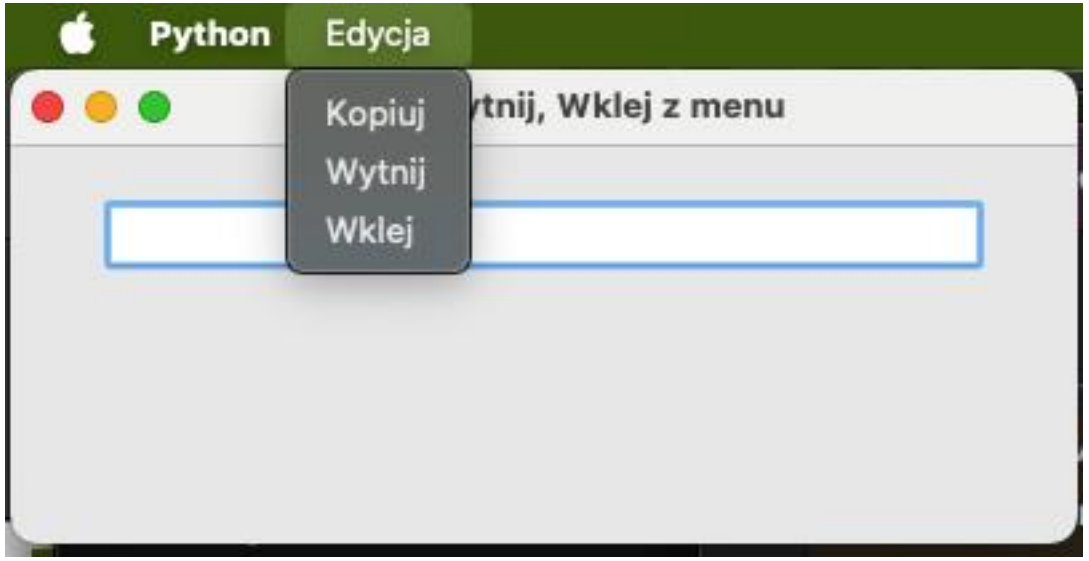


Uwaga!

Podgląd bazy danych z poziomu Pycharm jest możliwy po zainstalowaniu odpowiedniej wtyczki. Oczywiście alternatywą może być zwykły notatnik.

Dodatki?

Oczywiście menu z opcjami „wytnij”, „kopiuj” oraz „wklej”, dzięki któremu program będzie miał wiele różnych funkcjonalności.



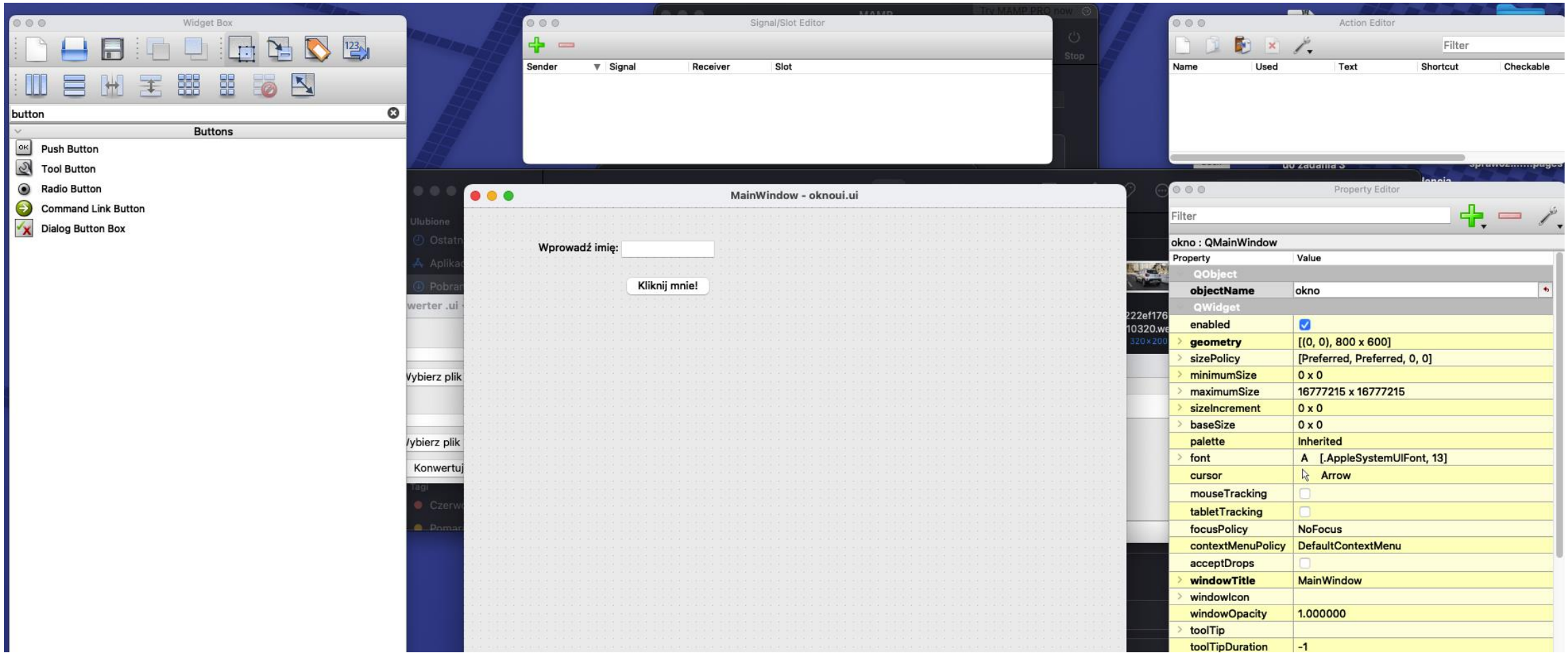

```
import tkinter as tk

class SchowekApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Kopiuj, Wytnij, Wklej z menu")
        self.root.geometry("400x150")

        # Pole tekstowe
        self.entry = tk.Entry(root, font=("Arial", 14), width=40)
        self.entry.pack(pady=20)

        # Menu
        self.menu = tk.Menu(root)
        root.config(menu=self.menu)

        edycja_menu = tk.Menu(self.menu, tearoff=0)

edycja_menu.add_command(label="Kopiuj", command=self.kopiuj)

edycja_menu.add_command(label="Wytnij", command=self.wytnij)

edycja_menu.add_command(label="Wklej", command=self.wklej)
        self.menu.add_cascade(label="Edycja", menu=edycja_menu)

    def kopiuj(self):
        self.entry.clipboard_clear()

self.entry.clipboard_append(self.entry.get())

    def wytnij(self):
        self.entry.clipboard_clear()

self.entry.clipboard_append(self.entry.get())
        self.entry.delete(0, 'end')

    def wklej(self):
        try:
            tekst = self.entry.selection_get(selection='CLIPBOARD')
            self.entry.insert('insert', tekst)
        except tk.TclError:
            pass  # Brak zawartości schowka

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

qT Designer – narzędzie do projektowania

Qt Designer to narzędzie do graficznego projektowania interfejsów użytkownika (GUI).

Jest częścią frameworka **Qt**, który służy do tworzenia aplikacji z graficznym interfejsem w C++, ale można go łatwo połączyć z Pythonem za pomocą biblioteki **PyQt5** lub **PySide2/PySide6**.

Umożliwia „przeciągnij i upuść” (drag & drop) komponentów jak przyciski, pola tekstowe, etykiety itp.


Widget Box
button
Buttons
Push Button
Tool Button
Radio Button
Command Link Button
Dialog Button Box
Signal/Slot Editor
Sender
Signal
Receiver
Slot
Action Editor
Filter
Name
Used
Text
Shortcut
Checkable
MainWindow - oknoui.ui
Wprowadź imię:
Kliknij mnie!
Property Editor
Filter
okno : QMainWindow
Property
Value
QObject
objectName
okno
QWidget
enabled
geometry
[(0, 0), 800 x 600]
sizePolicy
[Preferred, Preferred, 0, 0]
minimumSize
0 x 0
maximumSize
16777215 x 16777215
sizeIncrement
0 x 0
baseSize
0 x 0
palette
Inherited
font
A [.AppleSystemUIFont, 13]
cursor
Arrow
mouseTracking
tabletTracking
focusPolicy
NoFocus
contextMenuPolicy
DefaultContextMenu
acceptDrops
windowTitle
MainWindow
windowIcon
windowOpacity
1.000000
toolTip
toolTipDuration
-1

Obsługa qT Designer

Za pomocą kilku „kliknięć” tworzymy interfejs użytkownika. Dane zostaną zapisane w formacie „UI”. Będą one bezużyteczne – to tylko język opisowy, który stanie się częścią całego projektu.

Dlaczego tak?

Łatwiej i bezpieczniej jest zmienić wygląd niż cały kod. Mniejsze ryzyko uszkodzenia działających metod i funkcji. Nie musisz pisać wielu linii formatowania. Program sam to zrobi.

Jak to połączyć?

Python nie posiada natywnie Qt Designer, ale można go używać z:

- **PyQt5** (lub nowszym PyQt6)
- **PySide2/PySide6** – alternatywna wersja z licencją LGPL

Interfejsy projektujemy w Qt Designer i zapisujemy jako plik .ui

Plik .ui można:

- Załadować dynamicznie w Pythonie
- Lub przekonwertować do .py za pomocą pyuic5 (dla PyQt5)

Instalacja narzędzi do konwersji

Jest częścią pakietu **Qt**, ale można zainstalować przez pip:

pip install pyqt5-tools

Następnie:

pyqt5-tools

PyQt5:

pip install PyQt5

Polecenie w terminalu:

```
pyuic5 -o NAZWA_PLIKU.py NAZWA_PLIKU.ui
```

Uwaga!

W zależności od wersji Pythona PyQt i PySide można stosować zamiennie. Pełna lista na następnej stronie.

Tabela zależności pomiędzy Python a środowiskiem GUI

Wersja Pythona	PyQt5	PySide6
3.6	✅ działa	❌ brak wsparcia (PySide6 pojawiło się dopiero później)
3.7	✅ działa	✅ działa
3.8	✅ działa	✅ działa
3.9	✅ działa	✅ działa
3.10	✅ działa	✅ działa
3.11	✅ działa (ostatnia w pełni wspierana wersja)	✅ działa
3.12	⚠️ działa częściowo (trzeba nową wersję PyQt5, czasem są problemy)	✅ działa
3.13	❌ **brak wsparcia**	✅ działa **normalnie**

Różnice na przykładzie projektów

PyQt5	PySide6
``python	```python
from PyQt5 import QtWidgets	from PySide6 import QtWidgets
import sys	import sys
app = QtWidgets.QApplication(sys.argv)	app = QtWidgets.QApplication(sys.argv)
window = QtWidgets.QWidget()	window = QtWidgets.QWidget()
window.setWindowTitle("PyQt5")	window.setWindowTitle("PySide6")
button = QtWidgets.QPushButton("Kliknij mnie", window)	button = QtWidgets.QPushButton("Kliknij mnie", window)
button.move(50, 50)	button.move(50, 50)
window.show()	window.show()
sys.exit(app.exec_()) # PyQt5 używa exec_()	sys.exit(app.exec()) # PySide6 używa exec()
``	```

Różnice na przykładzie projektów

PyQt5	PySide6
``python	```python
from PyQt5 import QtWidgets, QtCore	from PySide6 import QtWidgets, QtCore
import sys	import sys
app = QtWidgets.QApplication(sys.argv)	app = QtWidgets.QApplication(sys.argv)
label = QtWidgets.QLabel("Witaj w PyQt5")	label = QtWidgets.QLabel("Witaj w PySide6")
label.resize(200, 100)	label.resize(200, 100)
label.setAlignment(QtCore.Qt.AlignCenter) # stary zapis	label.setAlignment(QtCore.Qt.AlignmentFlag.AlignCenter) # nowe enumy
label.show()	label.show()
sys.exit(app.exec_())	sys.exit(app.exec())
``	```

PyQt5

```python
from PyQt5 import QtWidgets, uic
import sys



class MyWindow(QtWidgets.QMainWindow):
def __init__(self):
super().__init__()
uic.loadUi("okno.ui", self) # proste ładowanie
self.button.clicked.connect(self.on_click)

def on_click(self):
QtWidgets.QMessageBox.information(self, "Info",
"Kliknięto przycisk!")



def on_click(self):
QtWidgets.QMessageBox.information(self, "Info",
"Kliknięto przycisk!")


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

PySide6

```python
from PySide6 import QtWidgets
from PySide6.QtUiTools import QUiLoader
from PySide6.QtCore import QFile
import sys


class MyWindow(QtWidgets.QMainWindow):
def __init__(self):
super().__init__()
file = QFile("okno.ui")
file.open(QFile.ReadOnly)
loader = QUiLoader()
self.ui = loader.load(file, self)
file.close()


self.ui.button.clicked.connect(self.on_click)


app = QtWidgets.QApplication(sys.argv)
window = MyWindow()
window.show()
sys.exit(app.exec())
```

Różnice w ładowaniu UI

Różnice:

1.PyQt5 – ma prostą funkcję uic.loadUi("plik.ui", self) → szybkie i wygodne.

2.PySide6 – trzeba użyć QUiLoader i QFile, albo zamiast tego wygenerować kod .py poleceniem:

3.pyside6-uic okno.ui -o okno.py

PySide6 – zastosowanie od Python 3.13

Podstawowe widgety

Widget	Opis
QMainWindow	Główne okno aplikacji, zawiera menu, pasek narzędzi, status itp.
QWidget	Podstawowa klasa bazowa dla wszystkich elementów GUI.
QDialog	Okno dialogowe (np. komunikat, formularz).
QFrame	Ramka (np. do grupowania elementów).
QGroupBox	Kontener z tytułem, grupujący widgety.
QTabWidget	Zakładki (np. karty w aplikacji).
QStackedWidget	Przełączanie między zestawami widgetów.
QScrollArea	Obszar przewijany dla dużych elementów.
QSplitter	Podział przestrzeni na panele.

Akcje, menu i paski

Widget	Opis
QPushButton	Przycisk.
QToolButton	Przycisk na pasku narzędzi.
QMenuBar, QMenu, QAction	Pasek menu i akcje.
QToolBar	Pasek narzędzi.
QStatusBar	Pasek stanu w QMainWindow.

PySide6 – zastosowanie od Python 3.13

Do wprowadzania i wyświetlania danych

Widget	Opis
QLineEdit	Pole tekstowe (jedna linia).
QTextEdit	Wieloliniowe pole tekstowe.
QPlainTextEdit	Prosty tekst bez formatowania.
QSpinBox	Wprowadzanie liczb całkowitych.
QDoubleSpinBox	Wprowadzanie liczb zmiennoprzecinkowych.
QComboBox	Lista rozwijana (menu wyboru).
QCheckBox	Pole wyboru (true/false).
QRadioButton	Przycisk radiowy (jeden z grupy).
QSlider	Suwak.
QDial	Pokrętło (jak głośność).
QDateEdit, QTimeEdit, QDateTimeEdit	Wybór daty i/lub godziny.
QFileDialog	Wybór pliku lub katalogu.
QColorDialog	Wybór koloru.

Widget	Opis
QLabel	Etykieta tekstowa lub graficzna.
QLCDNumber	Wyświetlacz numeryczny.
QProgressBar	Pasek postępu.
QTableWidget	Tabela z danymi.
QTreeWidget	Hierarchiczna lista (np. foldery).
QListWidget	Lista elementów.
QGraphicsView	Wyświetlanie sceny graficznej (2D).

Metody w PySide6

Klasy ogólne (QWidget, QMainWindow)

- show() — pokazuje okno
- hide() — ukrywa okno
- setWindowTitle("Tytuł") — ustawia tytuł okna
- setGeometry(x, y, width, height) — ustawia pozycję i rozmiar
- resize(width, height) — zmienia rozmiar
- move(x, y) — przenosi okno
- close() — zamyka okno

QLineEdit, QTextEdit

- setText("tekst") — ustawia tekst
- text() — pobiera tekst
- clear() — czyści pole
- setPlaceholderText("Wpisz...") — podpowiedź
- setReadOnly(True) — tryb tylko do odczytu

QLabel

- setText("napis") — ustawia tekst
- setPixmap(QPixmap("obraz.png")) — ustawia obraz
- setAlignment(Qt.AlignCenter) — wyrównanie

QSpinBox, QDoubleSpinBox

- setRange(min, max) — zakres wartości
- setValue(liczba) — ustawia wartość
- value() — pobiera wartość

QCheckBox

- isChecked() — sprawdza, czy zaznaczony
- setChecked(True/False) — zaznacza/odznacza
- stateChanged.connect(funkcja) — sygnał przy zmianie

QRadioButton

- isChecked() — sprawdza, czy wybrany
- setChecked(True) — zaznacza

QComboBox

- addItem("tekst") — dodaje opcję
- addItems(["a", "b"]) — dodaje wiele
- currentText() — pobiera aktualny tekst
- setCurrentIndex(i) — ustawia indeks

QTableWidget

- setRowCount(n) / setColumnCount(n) — ustawia wymiary
- setItem(row, col, QTableWidgetItem("tekst")) — dodaje komórkę
- item(row, col).text() — pobiera tekst
- insertRow(n) / removeRow(n) — zarządzanie wierszami

QMainWindow

- setCentralWidget(widget) — ustawia główny widok
- menuBar() — pobiera pasek menu
- statusBar() — pobiera pasek stanu
- addToolBar(toolbar) — dodaje pasek narzędzi

Widget	Sygnał	Argumenty	Opis
QPushButton	clicked	bool	Kliknięcie
QPushButton	pressed	—	Wciśnięcie
QPushButton	released	—	Zwolnienie
QPushButton	toggled	bool	Przełączenie
QLineEdit	textChanged	str	Zmiana tekstu
QLineEdit	textEdited	str	Edytowany przez użytkownika
QLineEdit	cursorPositionChanged	int,int	Ruch kursora
QLineEdit	editingFinished	—	ENTER/focus-out
QLineEdit	returnPressed	—	ENTER
QLineEdit	selectionChanged	—	Zmiana zaznaczenia
QTextEdit	textChanged	—	Zmiana tekstu
QTextEdit	cursorPositionChanged	—	Ruch kursora
QPlainTextEdit	textChanged	—	Zmiana tekstu
QPlainTextEdit	cursorPositionChanged	—	Ruch kursora
QCheckBox	stateChanged	int	0 (off) / 2 (on)
QCheckBox	toggled	bool	True/False
QRadioButton	toggled	bool	Zmiana wyboru
QRadioButton	clicked	bool	Kliknięcie
QComboBox	currentIndexChanged	int/str	Zmiana elementu
QComboBox	activated	int/str	Potwierdzony wybór
QComboBox	highlighted	int	Podświetlenie opcji
QSpinBox	valueChanged	int	Nowa wartość
QSpinBox	editingFinished	—	Koniec edycji
QDoubleSpinBox	valueChanged	double	Nowa wartość
QDoubleSpinBox	editingFinished	—	Koniec edycji
QSlider	valueChanged	int	Nowa pozycja
QSlider	sliderPressed	—	Start przesuwania
QSlider	sliderReleased	—	Koniec przesuwania
QSlider	sliderMoved	int	Ruch myszą
QSlider	rangeChanged	int,int	Zmiana zakresu
QDateEdit	dateChanged	QDate	Zmiana daty
QTimeEdit	timeChanged	QTime	Zmiana czasu
QDateTimeEdit	dateTimeChanged	QDateTime	Zmiana daty/czasu

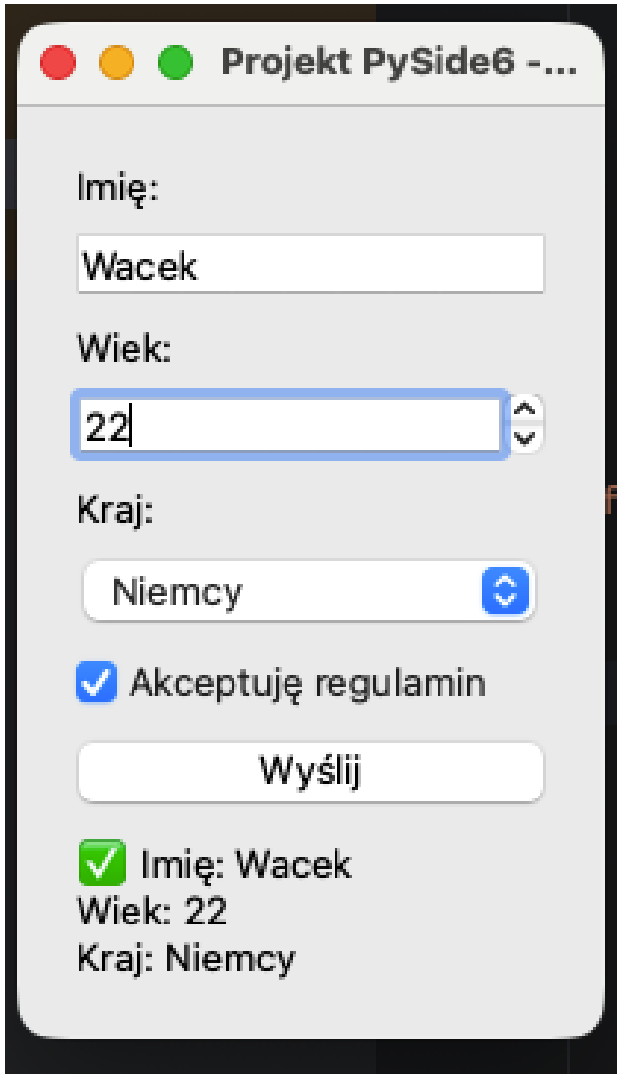
Sygnały – odpowiednik nasłuchiwacza

Widget	Sygnał	Argumenty	Opis
QListWidget	itemClicked	item*	Kliknięcie
QListWidget	itemDoubleClicked	item*	Dwuklik
QListWidget	itemSelectionChanged	—	Zmiana selekcji
QListWidget	currentItemChanged	item,item	Zmiana bieżącego
QTableWidget	cellClicked	int,int	Klik komórki
QTableWidget	cellActivated	int,int	Aktywacja
QTableWidget	cellChanged	int,int	Zmiana zawartości
QTableWidget	currentCellChanged	...	Nowa aktywna komórka
QTableWidget	itemChanged	item*	Edycja komórki
QTableWidget	itemSelectionChanged	—	Zmiana zaznaczenia
QTreeWidget	itemClicked	item,int	Klik
QTreeWidget	itemExpanded	item	Rozwinięcie
QTreeWidget	itemCollapsed	item	Zwinięcie
QTreeWidget	itemChanged	item,int	Edycja
QTabWidget	currentChanged	int	Zmiana zakładki
QTabWidget	tabCloseRequested	int	Żądanie zamknięcia
QTabWidget	tabBarClicked	int	Klik karty
QFileDialog	fileSelected	str	Wybrano plik
QFileDialog	filesSelected	list	Wiele plików
QFileDialog	directoryEntered	str	Wejście w katalog
QColorDialog	colorSelected	QColor	Wybrano kolor
QColorDialog	currentColorChanged	QColor	Podgląd
QProgressBar	valueChanged	int	Zmiana postępu
QAction	triggered	bool	Akcja wywołana
QAction	hovered	—	Najechanie
QAction	changed	—	Zmiana właściwości
QObject	destroyed	QObject*	Obiekt zniszczony
QObject	objectNameChanged	str	Zmiana nazwy

Sygnały – odpowiednik nasłuchiwacza

Przykładowy projekt z różnymi widgetami

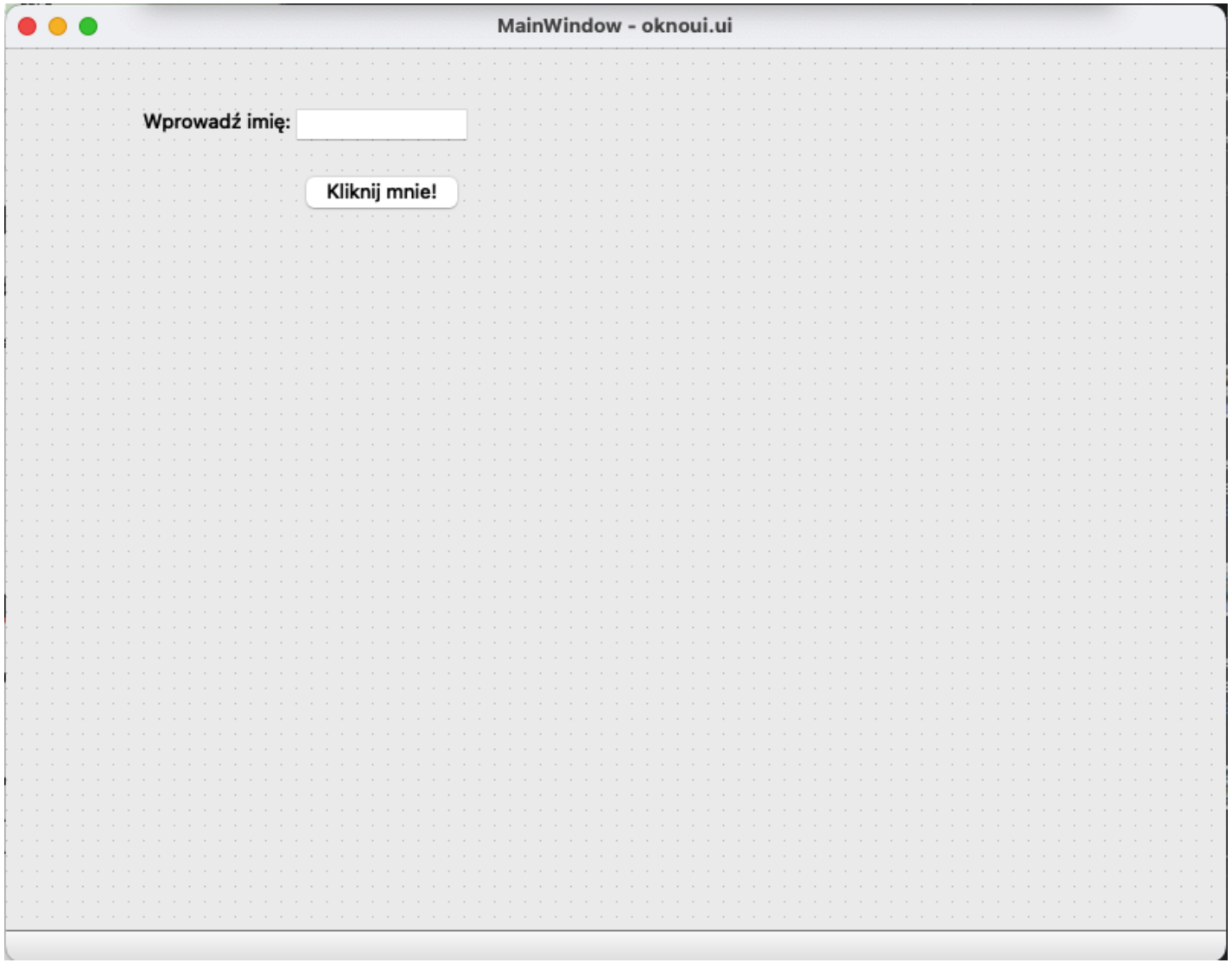
```
from PySide6.QtWidgets import (
    QApplication, QWidget, QLabel, QLineEdit, QSpinBox,
    QComboBox, QCheckBox, QPushButton, QVBoxLayout
)
import sys

class Okno(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Projekt PySide6 - pobieranie wartości")

        # --- Widgety ---
        self.label_imie = QLabel("Imię:")
        self.input_imie = QLineEdit()

        self.label_wiek = QLabel("Wiek:")
        self.input_wiek = QSpinBox()
        self.input_wiek.setRange(1, 120)

        self.label_kraj = QLabel("Kraj:")
        self.input_kraj = QComboBox()
        self.input_kraj.addItems(["Polska", "Niemcy", "USA", "UK",
"Czechy"])

        self.check_reg = QCheckBox("Akceptuję regulamin")

        self.button = QPushButton("Wyślij")
        self.output = QLabel("")

        # --- Layout ---
        layout = QVBoxLayout()
        layout.addWidget(self.label_imie)
        layout.addWidget(self.input_imie)

        layout.addWidget(self.label_wiek)
        layout.addWidget(self.input_wiek)

        layout.addWidget(self.label_kraj)
        layout.addWidget(self.input_kraj)

        layout.addWidget(self.check_reg)
        layout.addWidget(self.button)
        layout.addWidget(self.output)

        self.setLayout(layout)

        # --- Połączenie sygnału ---
        self.button.clicked.connect(self.pobierz_dane)

    # --- Funkcja zwracająca wartości z widgetów ---
    def pobierz_dane(self):
        imie = self.input_imie.text()
        wiek = self.input_wiek.value()
        kraj = self.input_kraj.currentText()
        regulamin = self.check_reg.isChecked()

        if not regulamin:
            self.output.setText("❌ Musisz zaakceptować regulamin!")
            return

        self.output.setText(
            f"✅ Imię: {imie}\n"
            f"Wiek: {wiek}\n"
            f"Kraj: {kraj}"
        )

# --- Start aplikacji ---
if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = Okno()
    window.show()
    sys.exit(app.exec())
```



Przykład prostego interfejsu

W Qt Designer:

- Tworzymy okno z:
  - QLabel: „Wprowadź imię”
  - QLineEdit
  - QPushButton: „Kliknij mnie”
- Zapisujemy jako okno.ui

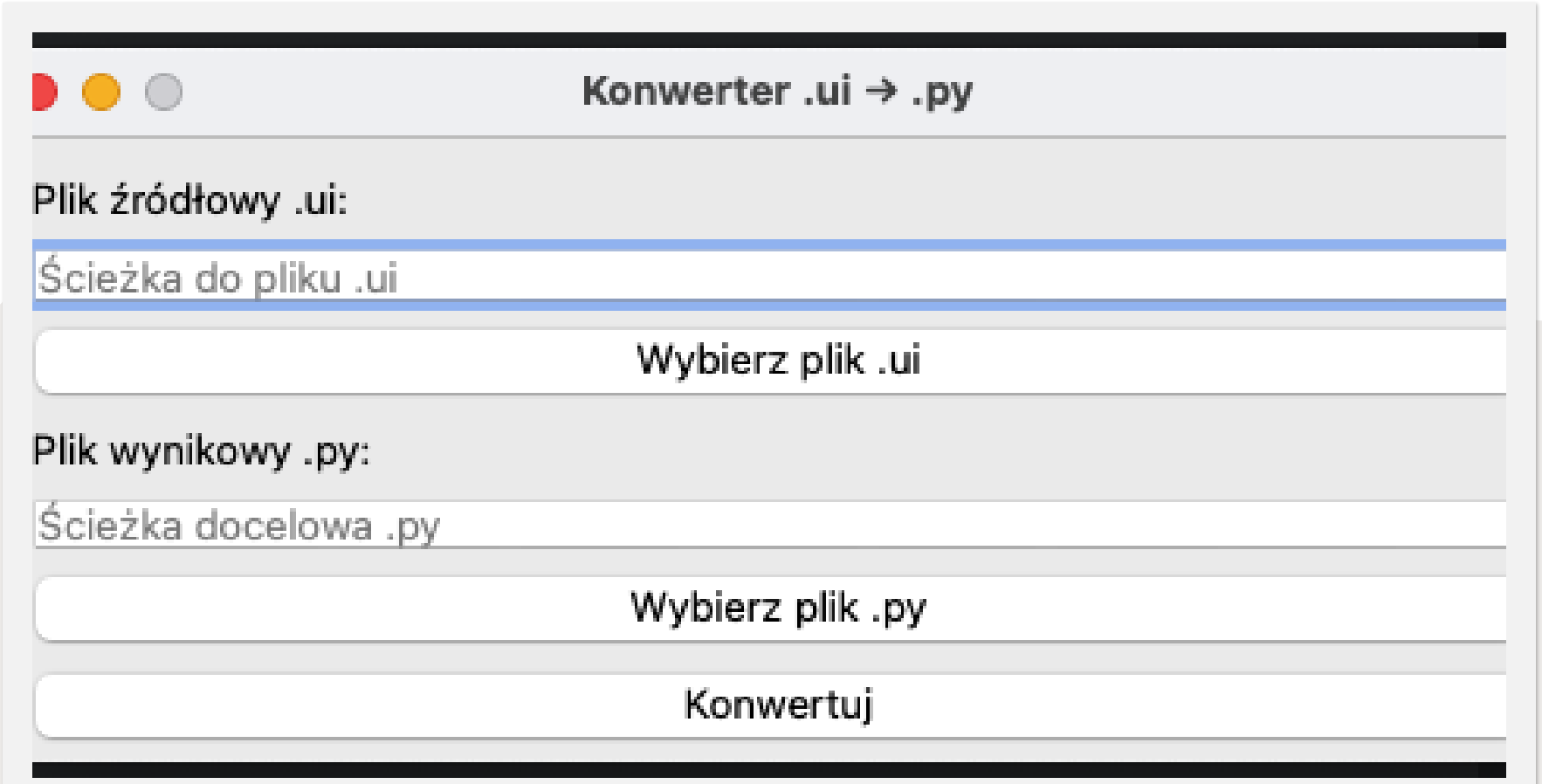


Bez konwersji ...

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>okno</class>
 <widget class="QMainWindow" name="okno">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>800</width>
    <height>600</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>MainWindow</string>
  </property>
  <widget class="QWidget" name="centralwidget">
   <widget class="QLabel" name="label">
    <property name="geometry">
     <rect>
      <x>90</x>
      <y>40</y>
      <width>101</width>
      <height>16</height>
     </rect>
    </property>
    <property name="text">
     <string>Wprowadź imię:</string>
    </property>
   </widget>
   <widget class="QLineEdit" name="lineEdit">
    <property name="geometry">
     <rect>
      <x>190</x>
      <y>40</y>
      <width>113</width>
      <height>21</height>
     </rect>
    </property>
   </widget>
   <widget class="QPushButton" name="pushButton">
    <property name="geometry">
     <rect>
      <x>190</x>
      <y>80</y>
      <width>113</width>
      <height>32</height>
     </rect>
    </property>
    <property name="text">
     <string>Kliknij mnie!</string>
    </property>
   </widget>
  </widget>
  <widget class="QStatusBar" name="statusbar"/>
 </widget>
 <resources/>
 <connections/>
</ui>
```

Wersja alternatywna – konwerter własnej produkcji

```
import sys
import subprocess
from pathlib import Path
from PyQt5.QtWidgets import (
    QApplication, QMainWindow, QFileDialog, QPushButton,
    QLabel, QLineEdit, QMessageBox, QWidget, QVBoxLayout,
QHBoxLayout
)

class UiConverter(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Konwerter .ui → .py")
        self.setFixedSize(500, 200)

        # Layout główny
        central_widget = QWidget()
        self.setCentralWidget(central_widget)
        layout = QVBoxLayout()

        # Wybór pliku .ui
        self.ui_path_input = QLineEdit()
        self.ui_path_input.setPlaceholderText("Ścieżka do pliku .ui")
        ui_button = QPushButton("Wybierz plik .ui")
        ui_button.clicked.connect(self.wybierz_ui)

        # Wybór pliku .py
        self.py_path_input = QLineEdit()
        self.py_path_input.setPlaceholderText("Ścieżka docelowa .py")
        py_button = QPushButton("Wybierz plik .py")
        py_button.clicked.connect(self.wybierz_py)

        # Przycisk Konwertuj
        convert_button = QPushButton("Konwertuj")
        convert_button.clicked.connect(self.konwertuj)

        # Dodaj do layoutu
        layout.addWidget(QLabel("Plik źródłowy .ui:"))
        layout.addWidget(self.ui_path_input)
        layout.addWidget(ui_button)

        layout.addWidget(QLabel("Plik wynikowy .py:"))
        layout.addWidget(self.py_path_input)
        layout.addWidget(py_button)

        layout.addWidget(convert_button)
        central_widget.setLayout(layout)

    def wybierz_ui(self):
        path, _ = QFileDialog.getOpenFileName(self, "Wybierz plik .ui", "",
"Pliki UI (*.ui)")
        if path:
            self.ui_path_input.setText(path)

    def wybierz_py(self):
        path, _ = QFileDialog.getSaveFileName(self, "Zapisz plik .py", "",
"Pliki Python (*.py)")
        if path:
            self.py_path_input.setText(path)

    def konwertuj(self):
        ui_path = self.ui_path_input.text()
        py_path = self.py_path_input.text()

        if not Path(ui_path).exists():
            QMessageBox.critical(self, "Błąd", "Plik .ui nie istnieje.")
            return

        if not py_path.endswith(".py"):
            QMessageBox.warning(self, "Uwaga", "Plik wynikowy powinien
mieć rozszerzenie .py")
            return

        try:
            subprocess.run(["pyuic5", "-o", py_path, ui_path], check=True)
            QMessageBox.information(self, "Sukces", f"Utworzono
plik:\n{py_path}")
        except FileNotFoundError:
            QMessageBox.critical(self, "Błąd", "Nie znaleziono narzędzia
pyuic5.\nZainstaluj je przez:\npip install pyqt5-tools")
        except subprocess.CalledProcessError:
            QMessageBox.critical(self, "Błąd", "Błąd podczas konwersji
pliku.")

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



Przydatne narzędzie

```
from PyQt5 import QtCore, QtGui, QtWidgets


class Ui_okno(object):
    def setupUi(self, okno):
        okno.setObjectName("okno")
        okno.resize(800, 600)
        self.centralwidget = QtWidgets.QWidget(okno)
        self.centralwidget.setObjectName("centralwidget")
        self.label = QtWidgets.QLabel(self.centralwidget)
        self.label.setGeometry(QtCore.QRect(90, 40, 101, 16))
        self.label.setObjectName("label")
        self.lineEdit = QtWidgets.QLineEdit(self.centralwidget)
        self.lineEdit.setGeometry(QtCore.QRect(190, 40, 113, 21))
        self.lineEdit.setObjectName("lineEdit")
        self.pushButton = QtWidgets.QPushButton(self.centralwidget)
        self.pushButton.setGeometry(QtCore.QRect(190, 80, 113, 32))
        self.pushButton.setObjectName("pushButton")
        okno.setCentralWidget(self.centralwidget)
        self.statusbar = QtWidgets.QStatusBar(okno)
        self.statusbar.setObjectName("statusbar")
        okno.setStatusBar(self.statusbar)

        self.retranslateUi(okno)
        QtCore.QMetaObject.connectSlotsByName(okno)

    def retranslateUi(self, okno):
        _translate = QtCore.QCoreApplication.translate
        okno.setWindowTitle(_translate("okno", "MainWindow"))
        self.label.setText(_translate("okno", "Wprowadź imię:"))
        self.pushButton.setText(_translate("okno", "Kliknij mnie!"))
```

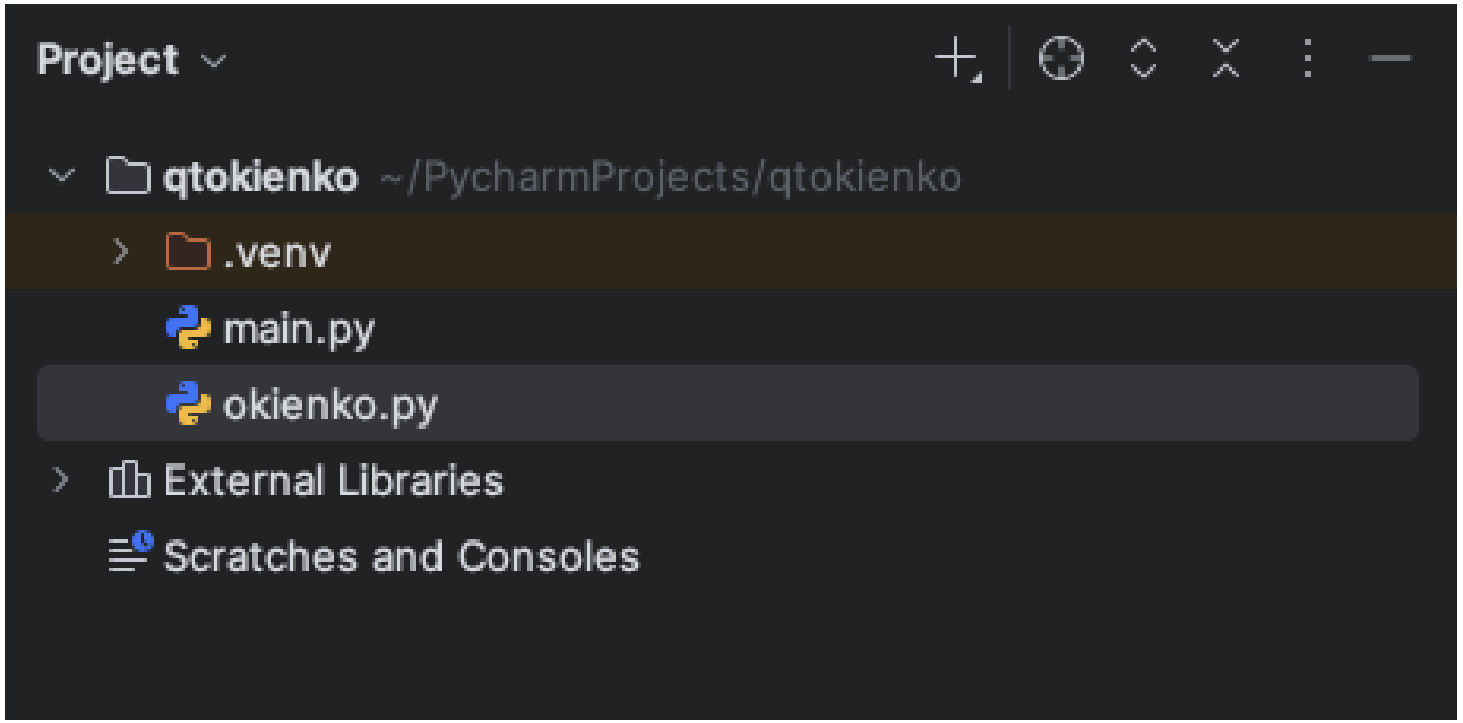
Z konwersją ...

```
import sys
from PyQt5.QtWidgets import QApplication,
QMainWindow
from okienko import Ui_okno  # import klasy z
wygenerowanego pliku

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.ui = Ui_okno()
        self.ui.setupUi(self)  # ustaw interfejs z pliku .ui

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

main.py

Dlaczego dwa pliki?

Utworzony przez wtyczkę plik okienko.py zawiera jedynie wygląd. Uruchomienie aplikacji odbywa się przez drugi plik, który zawiera importy, klasę i najważniejsze – konstruktor, który pomaga uruchomić program.



Prosta aplikacja z qT

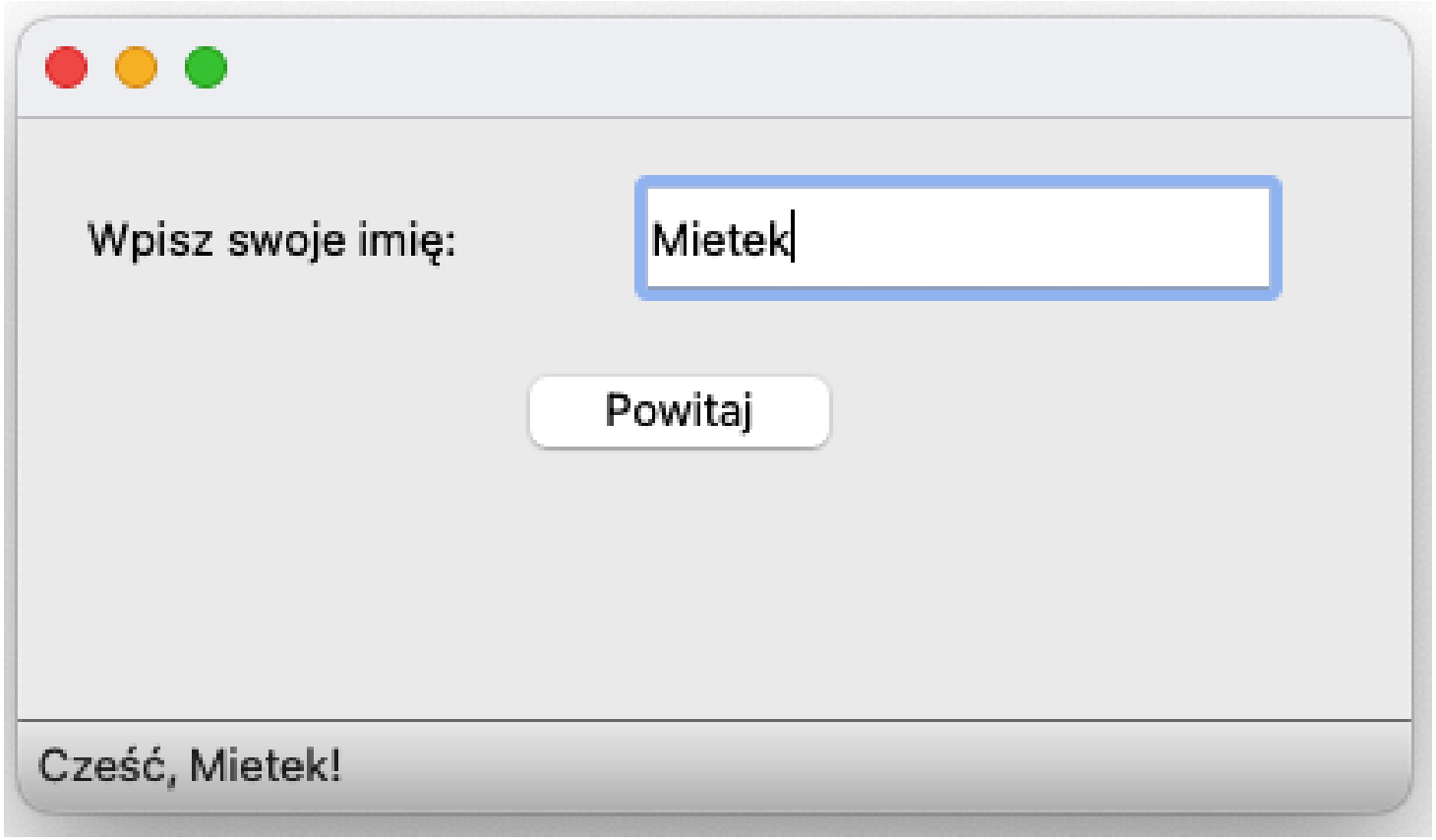
```
import sys
from PyQt5.QtWidgets import QApplication, QMainWindow
from okienko import Ui_okno  # importuj wygenerowany interfejs

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.ui = Ui_okno()
        self.ui.setupUi(self)

        # podłącz kliknięcie przycisku do funkcji
        self.ui.pushButton.clicked.connect(self.powitaj_uzytkownika)

    def powitaj_uzytkownika(self):
        imie = self.ui.lineEdit.text()
        self.ui.statusbar.showMessage(f"Cześć, {imie}!")

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

```
from PyQt5 import QtCore, QtGui, QtWidgets

class Ui_okno(object):
    def setupUi(self, okno):
        okno.setObjectName("okno")
        okno.resize(400, 200)
        self.centralwidget = QtWidgets.QWidget(okno)
        self.label = QtWidgets.QLabel("Wpisz swoje imię:",
self.centralwidget)
        self.label.setGeometry(QtCore.QRect(20, 20, 150, 30))
        self.lineEdit = QtWidgets.QLineEdit(self.centralwidget)
        self.lineEdit.setGeometry(QtCore.QRect(180, 20, 180, 30))
        self.pushButton = QtWidgets.QPushButton("Powitaj",
self.centralwidget)
        self.pushButton.setGeometry(QtCore.QRect(140, 70, 100,
30))
        self.statusbar = QtWidgets.QStatusBar(okno)
        okno.setCentralWidget(self.centralwidget)
        okno.setStatusBar(self.statusbar)
```

Efekt działania kodu

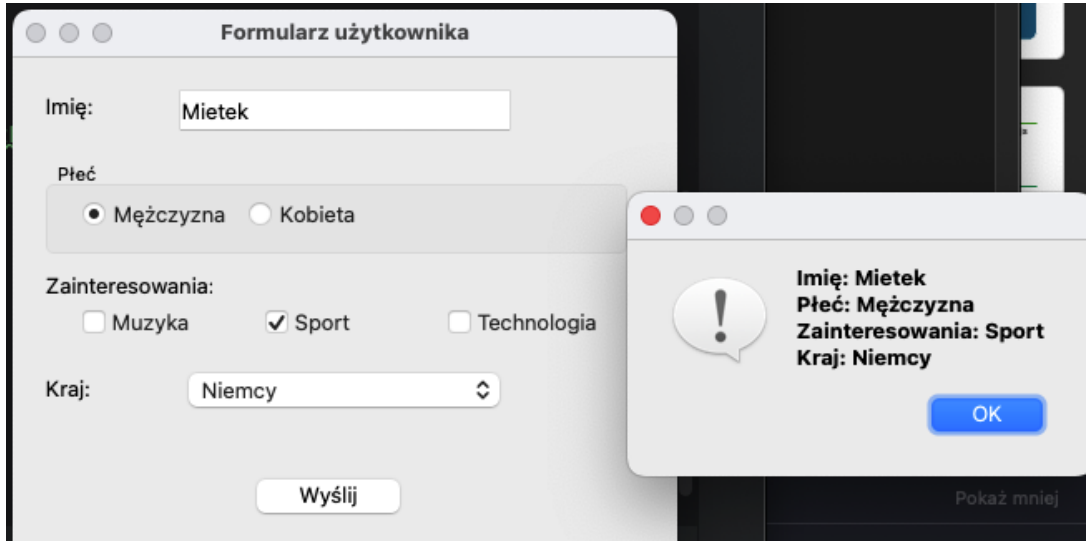
Aplikacja pobiera dane od użytkownika a następnie wyświetla je we wskazanym miejscu.

Należy pamiętać aby nie mieszać zawartości tych plików. Można to zrobić w jednym pliku, ale rozbicie projektu na wiele plików pozwala uzyskać logiczną strukturę.



Prosty program z różnymi widżetami

```
import sys
from PyQt5.QtWidgets import (
    QApplication, QMainWindow, QMessageBox
)
from formularz import Ui_okno  # import z wygenerowanego pliku

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.ui = Ui_okno()
        self.ui.setupUi(self)

        # Dodaj kraje do comboBox
        self.ui.comboCountry.addItems(["Polska", "Niemcy", "USA", "Japonia"])

        # Obsługa kliknięcia

self.ui.pushButtonSubmit.clicked.connect(self.pokaz_dane)

    def pokaz_dane(self):
        imie = self.ui.lineEditName.text()

        # Płeć
        if self.ui.radioMale.isChecked():
            plec = "Mężczyzna"
        elif self.ui.radioFemale.isChecked():
            plec = "Kobieta"
        else:
            plec = "Nie wybrano"

        # Zainteresowania
        zainteresowania = []
        if self.ui.checkMusic.isChecked():
            zainteresowania.append("Muzyka")
        if self.ui.checkSport.isChecked():
            zainteresowania.append("Sport")
        if self.ui.checkTech.isChecked():
            zainteresowania.append("Technologia")

        kraj = self.ui.comboCountry.currentText()

        dane = (
            f"Imię: {imie}\n"
            f"Płeć: {plec}\n"
            f"Zainteresowania: {', '.join(zainteresowania)}\n"
            f"Kraj: {kraj}"
        )

        QMessageBox.information(self, "Dane użytkownika", dane)

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



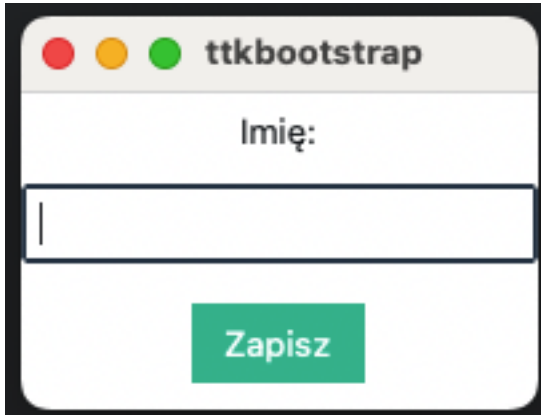
Prosty program z różnymi widżetami

```
from PyQt5 import QtCore, QtGui, QtWidgets


class Ui_okno(object):
    def setupUi(self, okno):
        okno.setObjectName("okno")
        okno.resize(400, 380)
        self.centralwidget = QtWidgets.QWidget(okno)
        self.centralwidget.setObjectName("centralwidget")
        self.labelName = QtWidgets.QLabel(self.centralwidget)
        self.labelName.setGeometry(QtCore.QRect(20, 20, 80, 20))
        self.labelName.setObjectName("labelName")
        self.lineEditName = QtWidgets.QLineEdit(self.centralwidget)
        self.lineEditName.setGeometry(QtCore.QRect(100, 20, 200, 25))
        self.lineEditName.setObjectName("lineEditName")
        self.groupBoxGender = QtWidgets.QGroupBox(self.centralwidget)
        self.groupBoxGender.setGeometry(QtCore.QRect(20, 60, 350, 60))
        self.groupBoxGender.setObjectName("groupBoxGender")
        self.radioMale = QtWidgets.QRadioButton(self.groupBoxGender)
        self.radioMale.setGeometry(QtCore.QRect(20, 25, 95, 20))
        self.radioMale.setObjectName("radioMale")
        self.radioFemale = QtWidgets.QRadioButton(self.groupBoxGender)
        self.radioFemale.setGeometry(QtCore.QRect(120, 25, 95, 20))
        self.radioFemale.setObjectName("radioFemale")
        self.labelHobbies = QtWidgets.QLabel(self.centralwidget)
        self.labelHobbies.setGeometry(QtCore.QRect(20, 130, 100, 16))
        self.labelHobbies.setObjectName("labelHobbies")
        self.checkMusic = QtWidgets.QCheckBox(self.centralwidget)
        self.checkMusic.setGeometry(QtCore.QRect(40, 150, 100, 20))
        self.checkMusic.setObjectName("checkMusic")
        self.checkSport = QtWidgets.QCheckBox(self.centralwidget)
        self.checkSport.setGeometry(QtCore.QRect(150, 150, 100, 20))
        self.checkSport.setObjectName("checkSport")
        self.checkTech = QtWidgets.QCheckBox(self.centralwidget)
        self.checkTech.setGeometry(QtCore.QRect(260, 150, 100, 20))
        self.checkTech.setObjectName("checkTech")
        self.labelCountry = QtWidgets.QLabel(self.centralwidget)
        self.labelCountry.setGeometry(QtCore.QRect(20, 190, 100, 20))
        self.labelCountry.setObjectName("labelCountry")
        self.comboCountry = QtWidgets.QComboBox(self.centralwidget)
        self.comboCountry.setGeometry(QtCore.QRect(100, 190, 200, 25))
        self.comboCountry.setObjectName("comboCountry")
        self.pushButtonSubmit = QtWidgets.QPushButton(self.centralwidget)
        self.pushButtonSubmit.setGeometry(QtCore.QRect(140, 250, 100, 32))
        self.pushButtonSubmit.setObjectName("pushButtonSubmit")
        okno.setCentralWidget(self.centralwidget)
        self.statusbar = QtWidgets.QStatusBar(okno)
        self.statusbar.setObjectName("statusbar")
        okno.setStatusBar(self.statusbar)

        self.retranslateUi(okno)
        QtCore.QMetaObject.connectSlotsByName(okno)

    def retranslateUi(self, okno):
        _translate = QtCore.QCoreApplication.translate
        okno.setWindowTitle(_translate("okno", "Formularz użytkownika"))
        self.labelName.setText(_translate("okno", "Imię:"))
        self.groupBoxGender.setTitle(_translate("okno", "Płeć"))
        self.radioMale.setText(_translate("okno", "Mężczyzna"))
        self.radioFemale.setText(_translate("okno", "Kobieta"))
        self.labelHobbies.setText(_translate("okno", "Zainteresowania:"))
        self.checkMusic.setText(_translate("okno", "Muzyka"))
        self.checkSport.setText(_translate("okno", "Sport"))
        self.checkTech.setText(_translate("okno", "Technologia"))
        self.labelCountry.setText(_translate("okno", "Kraj:"))
        self.pushButtonSubmit.setText(_translate("okno", "Wyślij"))
```

Material Design

pip install ttkbootstrap

```
import ttkbootstrap as ttk
from ttkbootstrap.constants import *

root = ttk.Window(themename="flatly")  # lub
"cosmo", "minty", "superhero", itp.

ttk.Label(root, text="Imię:").pack(pady=5)
ttk.Entry(root).pack(pady=5)
ttk.Button(root, text="Zapisz",
bootstyle="success").pack(pady=10)

root.mainloop()
```



Bootstrap w Python

W aplikacji webowej łączącej Python, HTML i CSS:

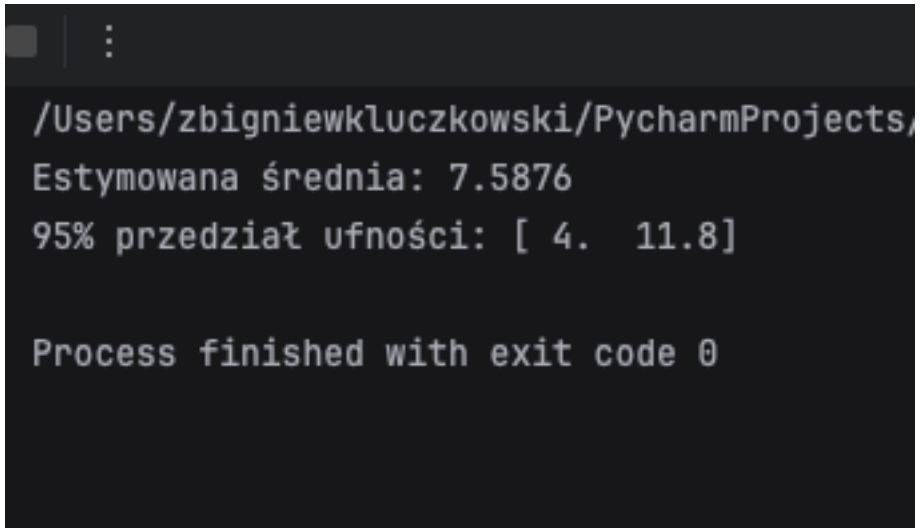
```
myapp/
 ├── app.py
 ├── templates/
 │   └── index.html
```

```
from flask import Flask, render_template

app = Flask(__name__)

@app.route('/')
def home():
    return render_template('index.html')
```

```
<!DOCTYPE html>
<html lang="pl">
<head>
    <meta charset="UTF-8">
    <title>Bootstrap w Flask</title>
    <!-- Bootstrap CDN -->
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/cs
s/bootstrap.min.css" rel="stylesheet">
</head>
<body class="bg-light">
    <div class="container mt-5">
        <h1 class="text-primary">Witaj w aplikacji Flask z
Bootstrapem!</h1>
        <button class="btn btn-success">Kliknij mnie</button>
    </div>
</body>
</html>
```

Bootstrap w Python

W połączeniu z Numpy do analizy danych:




```
import numpy as np

# Przykładowe dane
data = [2, 4, 7, 10, 15]

# Liczba prób bootstrapowych
n_bootstrap = 1000
means = []

for _ in range(n_bootstrap):
    sample = np.random.choice(data, size=len(data), replace=True)
    means.append(np.mean(sample))

# Średnia i przedział ufności
mean_estimate = np.mean(means)
conf_interval = np.percentile(means, [2.5, 97.5])

print(f"Estymowana średnia: {mean_estimate}")
print(f"95% przedział ufności: {conf_interval}")
```

Właściwości ttkbootstrap

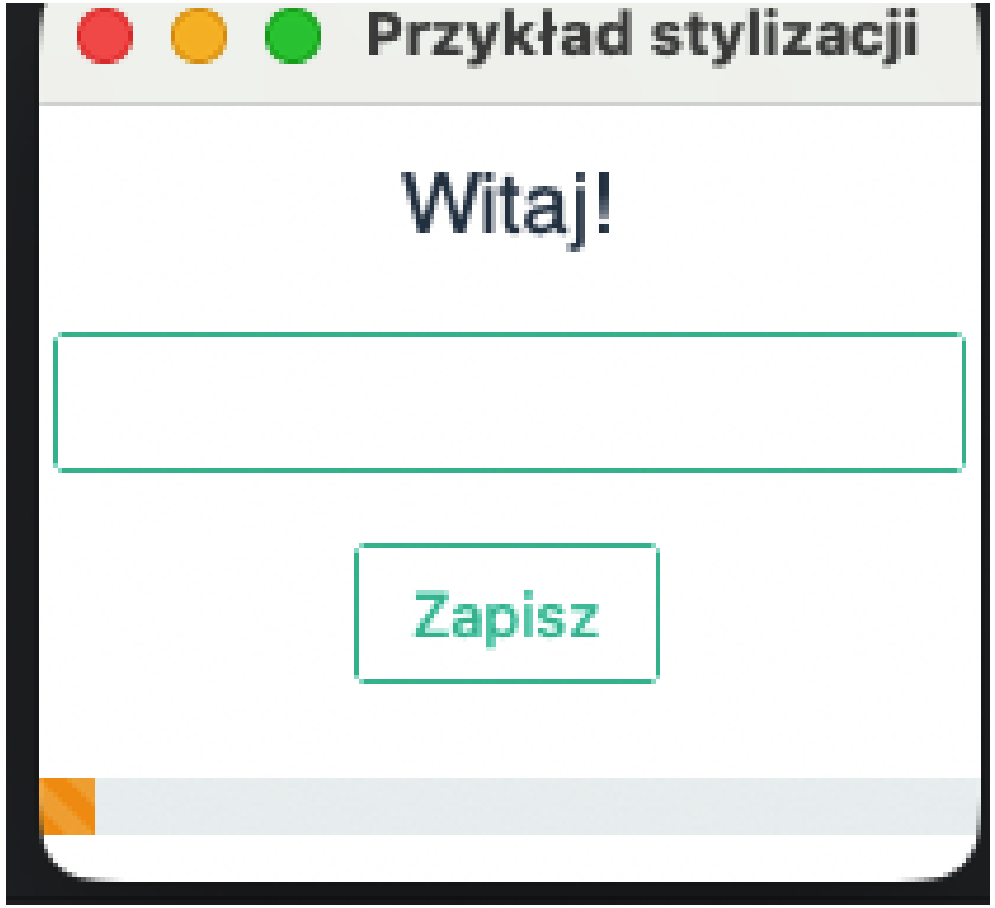
Widżet	Przykład stylu (bootstyle)	Efekt
Button	bootstyle="success"	Zielony przycisk Bootstrap
Button	bootstyle="warning-outline"	Żółty kontur, tło białe
Label	bootstyle="info"	Niebieski tekst
Entry	bootstyle="danger"	Czerwone obramowanie
Progressbar	bootstyle="primary-striped"	Paski z animacją
Frame	bootstyle="secondary"	Szare tło ramki
Checkbutton	bootstyle="success-toolbutton"	Zielony styl przycisku narzędziowego
Radiobutton	bootstyle="info"	Radiobutton z niebieskim stylem
Combobox	bootstyle="dark"	Ciemny styl rozwijanej listy
Notebook	bootstyle="primary"	Zakładki z niebieskim nagłówkiem

Popularne motywy

Nazwa motywu	Styl
flatly	Jasny, czysty, niebieski
cyborg	Ciemny, nowoczesny
superhero	Ciemny, kontrastowy
minty	Jasny, zielony akcent
darkly	Ciemny klasyczny
journal	Jasny, elegancki

```
import ttkbootstrap as ttk

app = ttk.Window(themename="superhero")
```

Przykład zastosowania

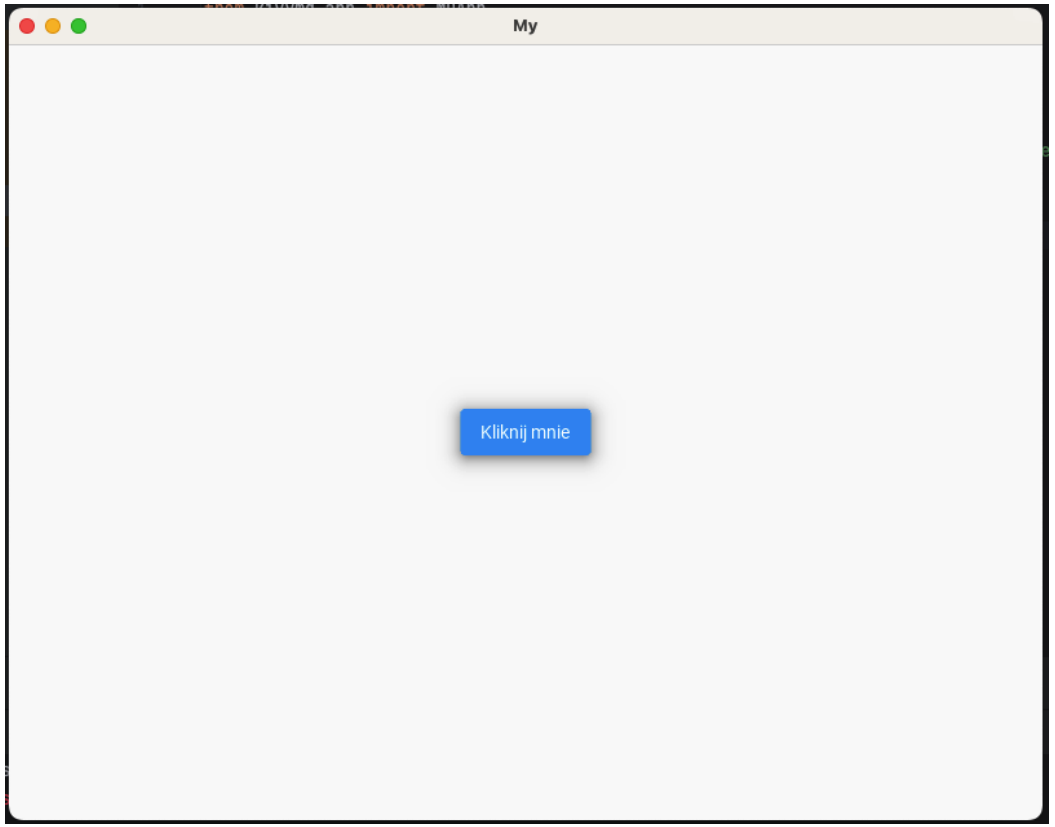



```
import ttkbootstrap as ttk

from ttkbootstrap.constants import *


app = ttk.Window(themename="flatly")

app.title("Przykład stylizacji")


ttk.Label(app, text="Witaj!", font=("Helvetica", 18),
bootstyle=PRIMARY).pack(pady=10)


ttk.Entry(app, bootstyle="success").pack(pady=5)


ttk.Button(app, text="Zapisz", bootstyle="success-
outline").pack(pady=10)


ttk.Progressbar(app, length=200, bootstyle="warning-striped",
mode="indeterminate").pack(pady=10)


app.mainloop()
```

Material Design

pip install kivy kivymd

```
from kivymd.app import MDApp
from kivymd.uix.button import MDRaisedButton

class MyApp(MDApp):
    def build(self):
        return MDRaisedButton(text="Kliknij mnie",
pos_hint={"center_x": 0.5, "center_y": 0.5})

MyApp().run()
```



Material Design

Obiekt	Właściwości Material Design	Odwzorowanie w Pythonie
Button	Zaokrąglone rogi, efekt falowania (ripple), cień (elevation), kolor primary lub accent	MDRaisedButton, ttk.Button z bootstyle
TextField	Pływający label (floating label), cienka linia pod spodem, walidacja, pomocniczy tekst	MDTextField, ttk.Entry z opisem i obramowaniem
Label	Lekkie, nowoczesne fonty (np. Roboto), typografia zgodna z hierarchią (subtitle1, body2)	MDLabel, ttk.Label z font=("Roboto", ...)
Card	Zaokrąglone krawędzie, cień, tło, padding, podział na nagłówek/treść	MDCard (KivyMD) lub Frame z tłem i borderem
Snackbar	Krótka wiadomość na dole ekranu, z opcjonalnym przyciskiem akcji	MDSnackbar (KivyMD)
Dropdown (Select)	Rozwijana lista z animacją, ikoną, etykietą pływającą	MDDropdownMenu, OptionMenu, ttk.Combobox
Switch / Toggle	Suwak z animacją, kolor zmieniający się po aktywacji	MDSwitch, stylizowany Checkbuttonlub ttk.Checkbutton
Dialog / Alert	Zaokrąglone okno, przezroczyste tło, ikony, przyciski akcji	MDDialog, messagebox (proste)
Toolbar / AppBar	Pasek górny z tytułem, ikoną i menu, kolory primary, elevation	MDTopAppBar, ręcznie tworzony Framez przyciskami
FAB (Floating Action Button)	Duży okrągły przycisk, unosi się, efekt cienia i ripple	MDFloatingActionButton, trudno odwzorować w Tkinter

Przykładowe właściwości w Material Design

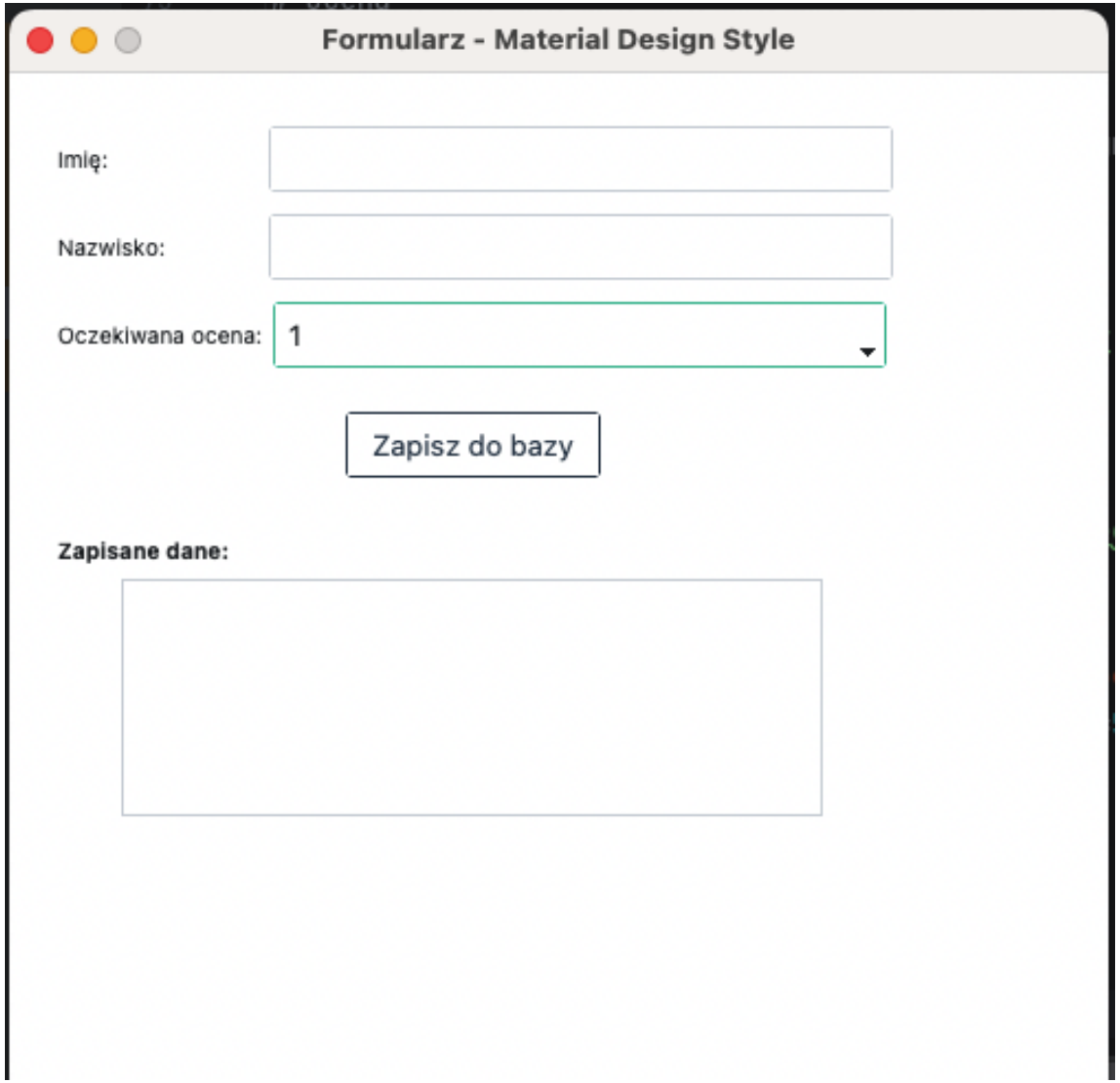
Atrybut	Przykład
Kolor główny	#6200EE (Deep Purple 500)
Kolor akcentu	#03DAC5 (Teal A200)
Cień (elevation)	1–24 (imituje uniesienie elementu)
Font podstawowy	Roboto, Noto, Arial, Segoe UI
Zaokrąglenie rogów	4dp, 8dp, 16dp
Padding	16dp, 24dp (dla kart, formularzy)
Ikony	Material Icons, np. menu, save

Zaawansowany przykład zastosowania

```
import sqlite3
import tkinter as tk
import ttkbootstrap as ttk
from ttkbootstrap.constants import *
from tkinter import messagebox

DB_PATH = "material_gui.db"

def init_db():
    conn = sqlite3.connect(DB_PATH)
    c = conn.cursor()
    c.execute('''
        CREATE TABLE IF NOT EXISTS users (
            id INTEGER PRIMARY KEY
AUTOINCREMENT,
            name TEXT NOT NULL,
            surname TEXT NOT NULL,
            grade INTEGER NOT NULL
        )
    ''')
    conn.commit()
    conn.close()

def save_to_db():
    name = entry_name.get().strip()
    surname = entry_surname.get().strip()
    grade = grade_var.get()

    if not name or not surname:
        messagebox.showerror("Błąd",
"Wypełnij wszystkie pola!")
        return

    conn = sqlite3.connect(DB_PATH)
    c = conn.cursor()
    c.execute("INSERT INTO users (name,
surname, grade) VALUES (?, ?, ?)", (name,
surname, grade))
    conn.commit()
    conn.close()

    entry_name.delete(0, tk.END)
    entry_surname.delete(0, tk.END)
    grade_var.set(1)

    refresh_data()

def refresh_data():
    listbox.delete(0, tk.END)
    conn = sqlite3.connect(DB_PATH)
    c = conn.cursor()
    c.execute("SELECT name, surname, grade
FROM users")
    for row in c.fetchall():
        listbox.insert(tk.END, f"{row[0]} {row[1]} -
ocena: {row[2]}")
    conn.close()

# --- Główne okno ---
root = ttk.Window(themename="flatly")
root.title("Formularz - Material Design Style")
root.geometry("500x470")
root.resizable(False, False)

# --- Formularz ---
frame = ttk.Frame(root, padding=20)
frame.pack(fill=tk.BOTH, expand=True)

# Imię
ttk.Label(frame, text="Imię:", font=("Segoe
UI", 10)).grid(row=0, column=0, sticky=tk.W,
pady=5)
entry_name = ttk.Entry(frame, width=30)
entry_name.grid(row=0, column=1, pady=5)

# Nazwisko
ttk.Label(frame, text="Nazwisko:",
font=("Segoe UI", 10)).grid(row=1, column=0,
sticky=tk.W, pady=5)
entry_surname = ttk.Entry(frame, width=30)
entry_surname.grid(row=1, column=1,
pady=5)

# Ocena
ttk.Label(frame, text="Oczekiwana ocena:",
font=("Segoe UI", 10)).grid(row=2, column=0,
sticky=tk.W, pady=5)
grade_var = ttk.IntVar(value=1)
select_grade = ttk.Combobox(frame,
textvariable=grade_var, values=[1, 2, 3, 4, 5],
width=28, bootstyle="success")
select_grade.grid(row=2, column=1, pady=5)

# Przycisk zapisu
btn_save = ttk.Button(frame, text="Zapisz do
bazy", command=save_to_db,
bootstyle="primary-outline")
btn_save.grid(row=3, columnspan=2,
pady=15)

# Lista wyników
ttk.Label(frame, text="Zapisane dane:",
font=("Segoe UI", 10, "bold")).grid(row=4,
column=0, columnspan=2, sticky=tk.W,
pady=(10, 0))

# Użycie klasycznego Listbox z tkinter
listbox = tk.Listbox(frame, height=8,
width=45, bg="#e8f0fe", fg="#000000",
font=("Segoe UI", 10))
listbox.grid(row=5, column=0,
columnspan=2, pady=5)

# Baza i uruchomienie
init_db()
refresh_data()
root.mainloop()
```

Efekt?

Prosty formularz z zapisem do bazy danych z podświetleniem pola do wprowadzania danych.



Nietypowe formatowania

Zaokrąglone okno:

```
root.overrideredirect(True)  # Ukrycie ramki systemowej

# Można potem dodać własne "ramki" i przyciski zamykania
```

Zmiana kursora myszy:

```
root.config(cursor="hand2")  # Przykłady: "watch", "cross", "xterm", "heart", "pirate"
```

Przezroczystość:

```
root.attributes("-alpha", 0.9)  # Od 0 (całkowicie przezroczyste) do 1 (pełna widoczność)
```

Okno zawsze na wierzchu:

```
root.attributes("-topmost", True)  # Okno nad wszystkimi innymi
```

Animowane okno:

```
def fade_in():
    alpha = root.attributes("-alpha")
    if alpha < 1:
        alpha += 0.05
        root.attributes("-alpha", alpha)
        root.after(30, fade_in)

root.attributes("-alpha", 0.0)
fade_in()
```

Nietypowe formatowania

Zmiana tytułu w czasie rzeczywistym:

```
def update_title():
    import datetime
    now = datetime.datetime.now().strftime("%H:%M:%S")
    root.title(f"Aktualny czas: {now}")
    root.after(1000, update_title)


update_title()
```

Przyciski na dole okna:

```
import tkinter as tk

root = tk.Tk()
root.geometry("400x300")

# Główna zawartość
main_frame = tk.Frame(root)
main_frame.pack(fill='both', expand=True)

# Dolne "menu"
bottom_menu = tk.Frame(root, bg="#dddddd", height=40)
bottom_menu.pack(side='bottom', fill='x')

# Przykładowe przyciski w menu
tk.Button(bottom_menu, text="Start").pack(side="left", padx=10,
pady=5)
tk.Button(bottom_menu, text="Ustawienia").pack(side="left", padx=10,
pady=5)
tk.Button(bottom_menu, text="Wyjście",
command=root.quit).pack(side="right", padx=10, pady=5)

root.mainloop()
```

Prezentacja danych w Pythonie

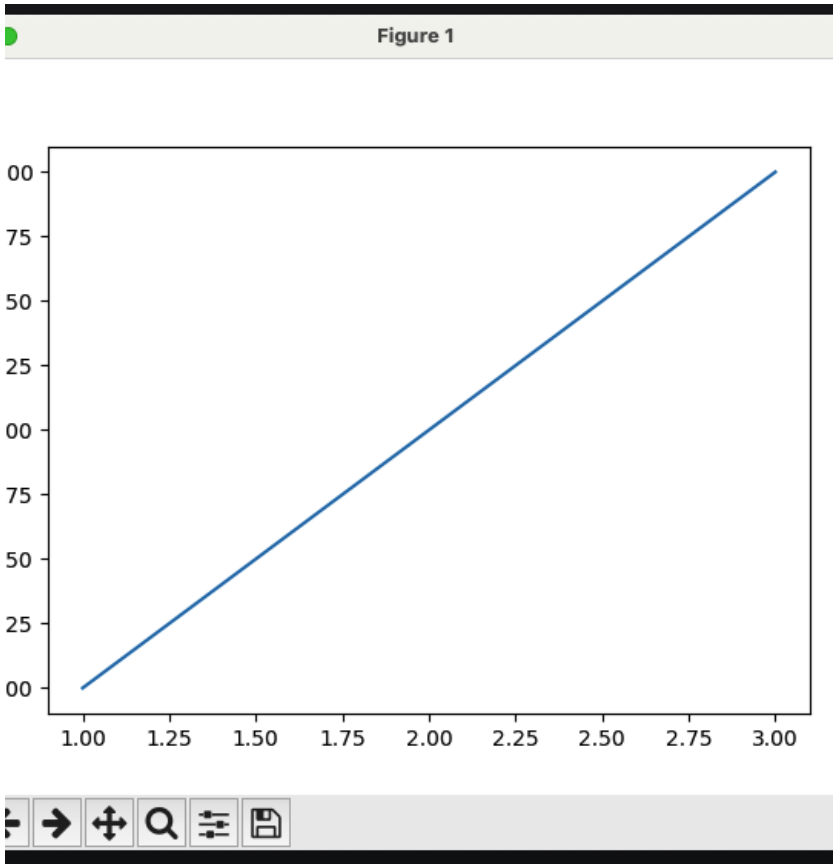
Matplotlib to jedna z najpopularniejszych bibliotek w Pythonie do tworzenia **wizualizacji danych**. Umożliwia generowanie różnych typów wykresów: liniowych, słupkowych, kołowych, histogramów, wykresów 3D i wielu innych.

- Obsługuje eksport wykresów do plików (PNG, PDF, SVG itp.).
- Jest szeroko wykorzystywana w analizie danych, nauce i inżynierii.
- Może być używana razem z bibliotekami NumPy, Pandas, SciPy itp.

Matplotlib.pyplot

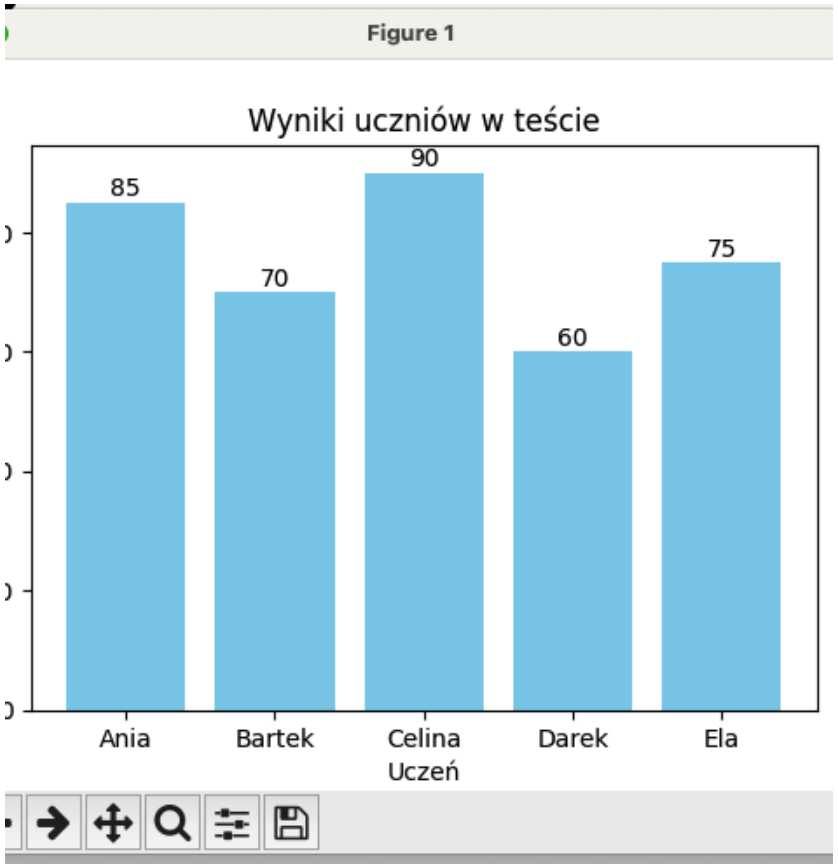
- pyplot to **moduł** w bibliotece matplotlib, który dostarcza prosty interfejs podobny do funkcji z MATLAB-a. Pozwala tworzyć wykresy w sposób sekwencyjny (krok po kroku) – np. tworzenie wykresu liniowego to:

```
import matplotlib.pyplot as plt
plt.plot([1, 2, 3], [4, 5, 6])
plt.show()
```



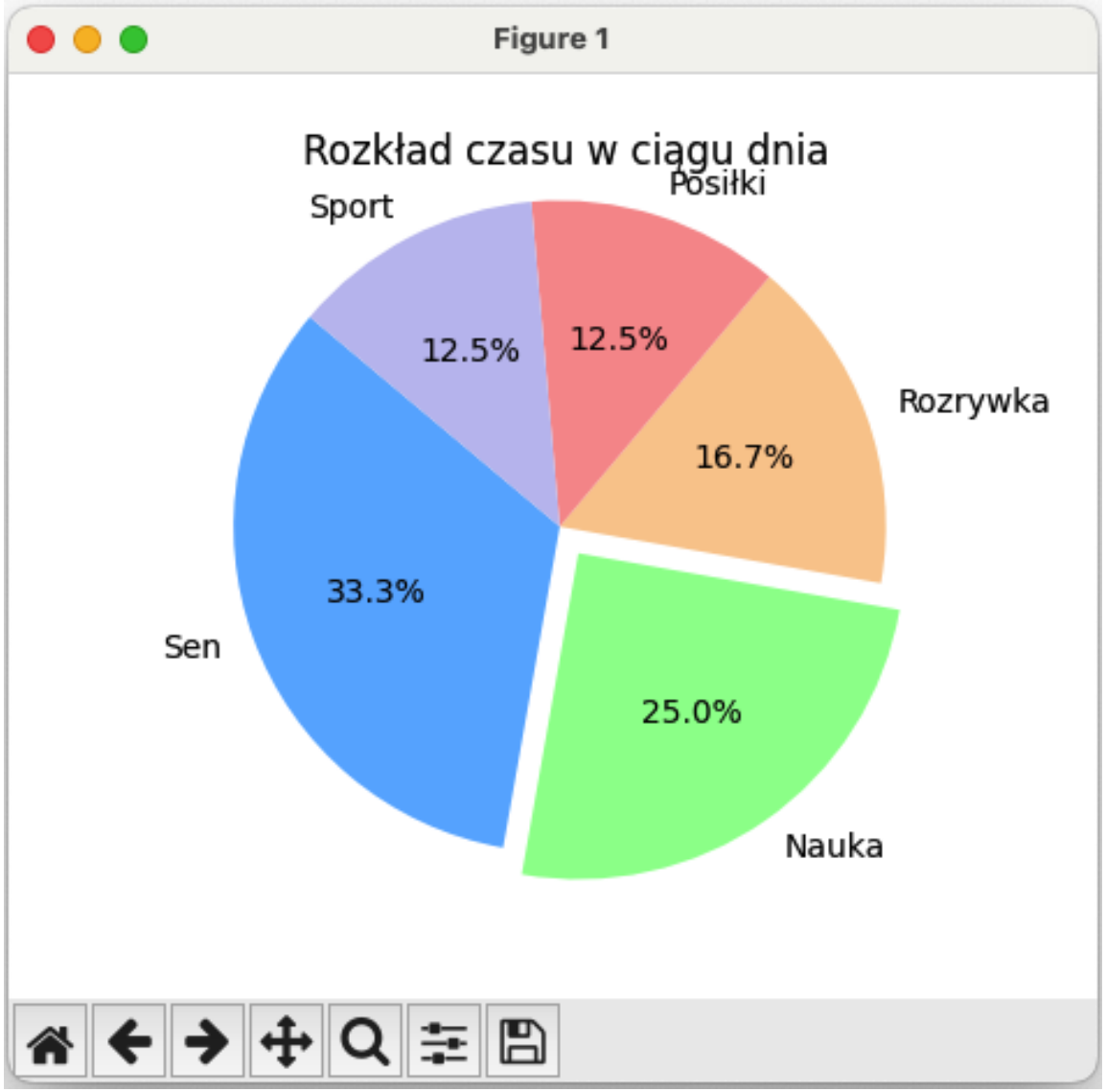
Prezentacja danych w Pythonie

```
import matplotlib.pyplot as plt

uczniowie = ['Ania', 'Bartek', 'Celina', 'Darek', 'Ela']
punkty = [85, 70, 90, 60, 75]

plt.bar(uczniowie, punkty, color='skyblue')
plt.title('Wyniki uczniów w teście')
plt.xlabel('Uczeń')
plt.ylabel('Punkty')

# Dodanie wartości nad słupkami
for i in range(len(punkty)):
    plt.text(i, punkty[i] + 1, str(punkty[i]), ha='center')

plt.show()
```

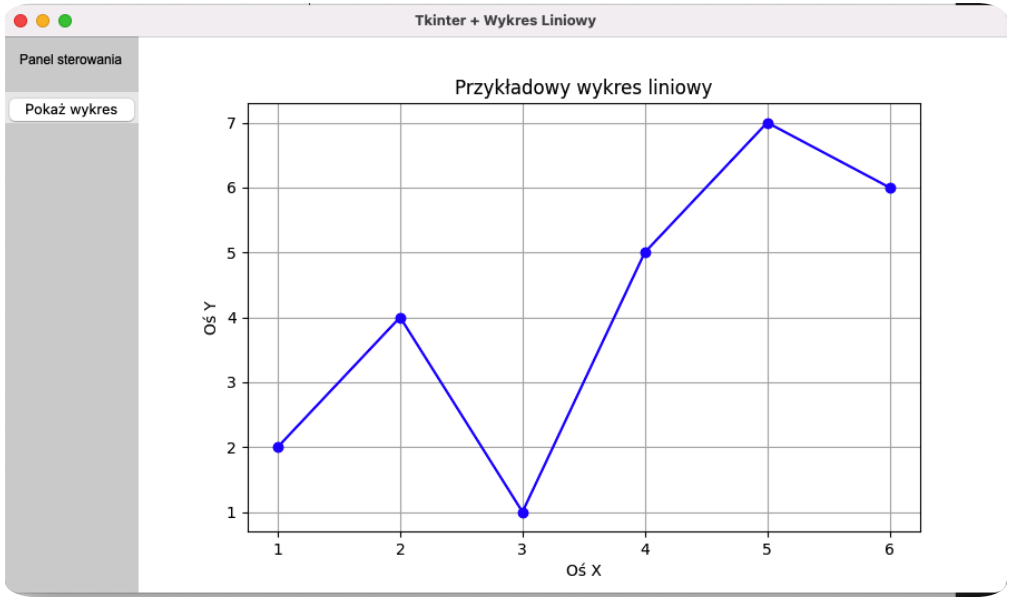


Prezentacja danych w Pythonie

```
import matplotlib.pyplot as plt

czynnosci = ['Sen', 'Nauka', 'Rozrywka', 'Posiłki', 'Sport']
godziny = [8, 6, 4, 3, 3]
kolory = ['#66b3ff', '#99ff99', '#ffcc99', '#ff9999', '#c2c2f0']
explode = [0, 0.1, 0, 0, 0]  # Wyróżnienie nauki

plt.pie(godziny, labels=czynnosci, colors=kolory,
autopct='%1.1f%%', startangle=140, explode=explode)
plt.title('Rozkład czasu w ciągu dnia')
plt.axis('equal')  # Równe proporcje
plt.show()
```



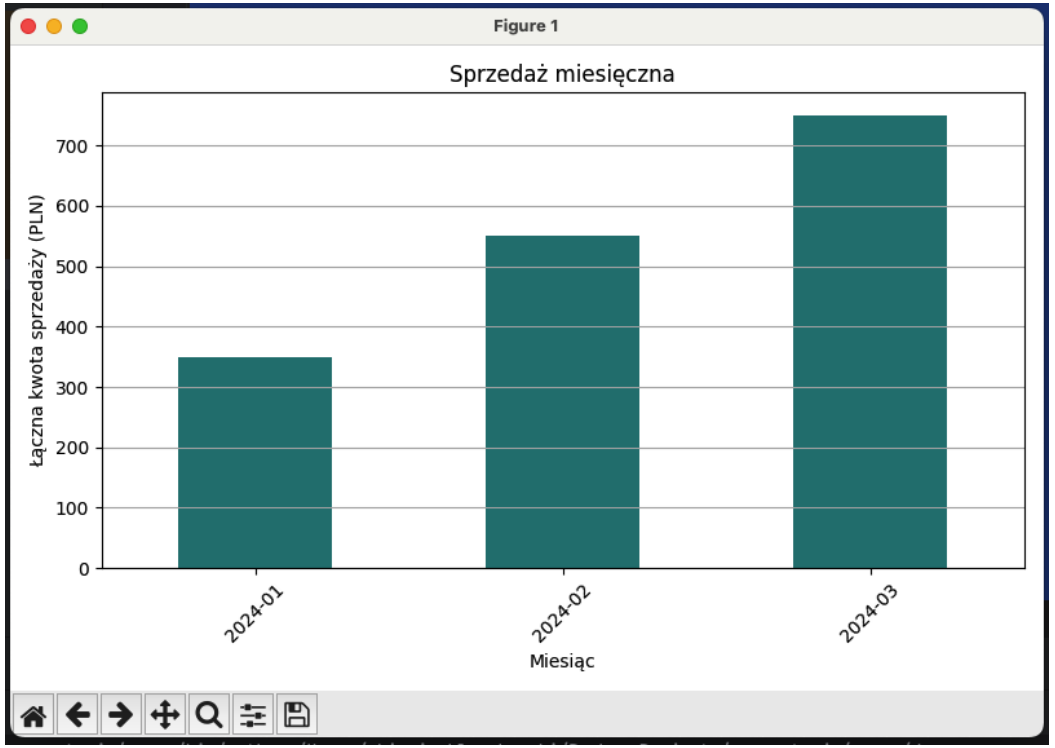
Prezentacja danych w Pythonie




```
import tkinter as tk
from tkinter import ttk
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg

# --- Przykładowe dane ---
dane_x = [1, 2, 3, 4, 5, 6]
dane_y = [2, 4, 1, 5, 7, 6]

# --- Funkcja do tworzenia wykresu ---
def pokaz_wykres():
    fig = plt.Figure(figsize=(5, 4), dpi=100)
    ax = fig.add_subplot(111)
    ax.plot(dane_x, dane_y, marker='o', color='blue')
    ax.set_title('Przykładowy wykres liniowy')
    ax.set_xlabel('Oś X')
    ax.set_ylabel('Oś Y')
    ax.grid(True)

    canvas = FigureCanvasTkAgg(fig, master=panel_wykresu)
    canvas.draw()
    canvas.get_tk_widget().pack(fill=tk.BOTH, expand=True)

# --- GUI ---
root = tk.Tk()
root.title("Tkinter + Wykres Liniowy")
root.geometry("900x500")

# --- Lewy panel: kontrolki ---
panel_lewy = tk.Frame(root, width=200, bg="lightgray")
panel_lewy.pack(side="left", fill="y")

tk.Label(panel_lewy, text="Panel sterowania", font=("Arial", 12), bg="lightgray").pack(pady=10)
tk.Button(panel_lewy, text="Pokaż wykres", command=pokaz_wykres).pack(pady=10)

# --- Prawy panel: wykres ---
panel_wykresu = tk.Frame(root, bg="white")
panel_wykresu.pack(side="right", fill="both", expand=True)

# --- Uruchomienie ---
root.mainloop()
```

Prezentacja danych w Pythonie

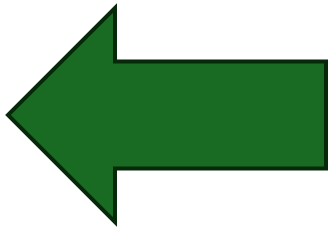
Dane pobierane z pliku możemy zaprezentować za pomocą biblioteki Pandas.




```
import pandas as pd
import matplotlib.pyplot as plt

# Wczytanie danych z pliku
df = pd.read_csv('sprzedaz.csv', parse_dates=['Data'])

# Dodanie kolumny z nazwą miesiąca
df['Miesiac'] = df['Data'].dt.strftime('%Y-%m')

# Grupowanie danych i sumowanie sprzedaży
sprzedaz_miesieczna = df.groupby('Miesiac')['Kwota'].sum()

# Wykres słupkowy
plt.figure(figsize=(8, 5))
sprzedaz_miesieczna.plot(kind='bar', color='teal')

plt.title('Sprzedaż miesięczna')
plt.xlabel('Miesiąc')
plt.ylabel('Łączna kwota sprzedaży (PLN)')
plt.grid(axis='y')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

Prezentacja danych w Pythonie

```
Data,Kwota
2024-01-15,150
2024-01-22,200
2024-02-01,300
2024-02-18,250
2024-03-05,400
2024-03-19,350
```



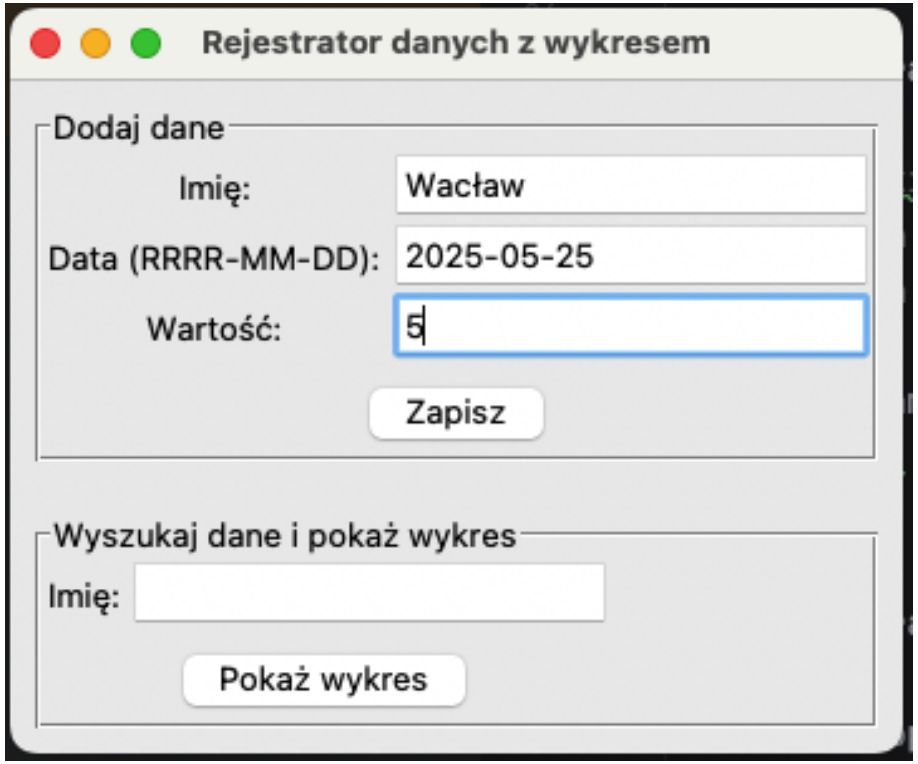
Plik CSV przechowuje dane. Muszą być odpowiednio posortowane. Plik Pythona otwiera plik z danymi a Pandas pozwala na interpretację i wyświetlenie danych na wykresie.

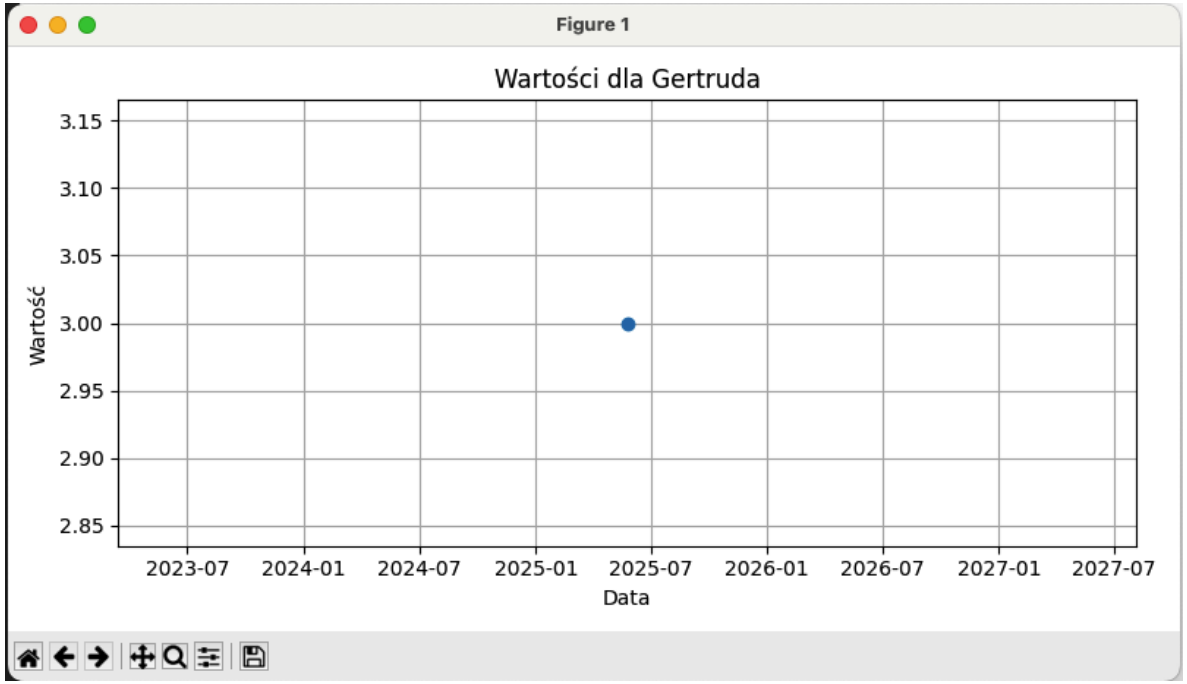
Pandas zawsze jest powiązany z Matplotlib. Tylko tak można zapewnić prawidłowe działanie programu. Oczywiście można napisać kod, który jednocześnie pozwala wprowadzić dane w formie formularza, zapisuje je w pliku CSV i pozwala wyszukać dane i je zaprezentować.

Prezentacja danych w Pythonie

```
import tkinter as tk
from tkinter import messagebox
import pandas as pd
import matplotlib.pyplot as plt
from datetime import datetime

PLIK_CSV = "dane.csv"

# --- Funkcje pomocnicze ---

def zapisz_dane():
    imie = entry_imie.get().strip()
    data = entry_data.get().strip()
    wartosc = entry_wartosc.get().strip()

    if not imie or not data or not wartosc:
        messagebox.showerror("Błąd", "Wszystkie pola muszą być wypełnione.")
        return

    try:
        data_parsed = datetime.strptime(data, "%Y-%m-%d")
        wartosc_float = float(wartosc)
    except:
        messagebox.showerror("Błąd", "Niepoprawny format daty (RRRR-MM-DD) lub liczby.")
        return

    nowy_wiersz = pd.DataFrame([[imie, data_parsed, wartosc_float]], columns=["Imię", "Data", "Wartość"])

    try:
        istnieje = pd.read_csv(PLIK_CSV)
        nowy_wiersz.to_csv(PLIK_CSV, mode='a', index=False, header=False)
    except FileNotFoundError:
        nowy_wiersz.to_csv(PLIK_CSV, index=False)

    messagebox.showinfo("Sukces", "Dane zapisane!")
    entry_imie.delete(0, tk.END)
    entry_data.delete(0, tk.END)
    entry_wartosc.delete(0, tk.END)

def wyszukaj_i_wyswietl():
    imie_szukane = entry_szukaj.get().strip()

    try:
        df = pd.read_csv(PLIK_CSV, parse_dates=["Data"])
    except FileNotFoundError:
        messagebox.showerror("Błąd", "Brak pliku z danymi.")
        return

    df_osoba = df[df["Imię"].str.lower() == imie_szukane.lower()]

    if df_osoba.empty:
        messagebox.showinfo("Brak danych", f"Brak danych dla {imie_szukane}.")
        return

    df_osoba = df_osoba.sort_values("Data")

    plt.figure(figsize=(8, 4))
    plt.plot(df_osoba["Data"], df_osoba["Wartość"], marker='o')
    plt.title(f"Wartości dla {imie_szukane}")
    plt.xlabel("Data")
    plt.ylabel("Wartość")
    plt.grid(True)
    plt.tight_layout()
    plt.show()

# --- Interfejs graficzny (Tkinter) ---

root = tk.Tk()
root.title("Rejestrator danych z wykresem")

# --- Formularz dodawania danych ---
frame_form = tk.LabelFrame(root, text="Dodaj dane")
frame_form.pack(padx=10, pady=10, fill="x")

tk.Label(frame_form, text="Imię:").grid(row=0, column=0)
tk.Label(frame_form, text="Data (RRRR-MM-DD):").grid(row=1, column=0)
tk.Label(frame_form, text="Wartość:").grid(row=2, column=0)

entry_imie = tk.Entry(frame_form)
entry_data = tk.Entry(frame_form)
entry_wartosc = tk.Entry(frame_form)

entry_imie.grid(row=0, column=1)
entry_data.grid(row=1, column=1)
entry_wartosc.grid(row=2, column=1)

tk.Button(frame_form, text="Zapisz", command=zapisz_dane).grid(row=3, columnspan=2, pady=5)

# --- Wyszukiwanie i wykres ---
frame_search = tk.LabelFrame(root, text="Wyszukaj dane i pokaż wykres")
frame_search.pack(padx=10, pady=10, fill="x")

tk.Label(frame_search, text="Imię:").grid(row=0, column=0)
entry_szukaj = tk.Entry(frame_search)
entry_szukaj.grid(row=0, column=1)

tk.Button(frame_search, text="Pokaż wykres", command=wyszukaj_i_wyswietl).grid(row=1, columnspan=2, pady=5)

root.mainloop()
```

Prezentacja danych w Pythonie





Prezentacja danych w Pythonie

Powyższy program poprzez formularz tworzy strukturę danych, która zapisuje się w pliku CSV. Następnie wyszukiwarka pozwala odczytać dane przypisane do konkretnej osoby i wyświetla je na wykresie.

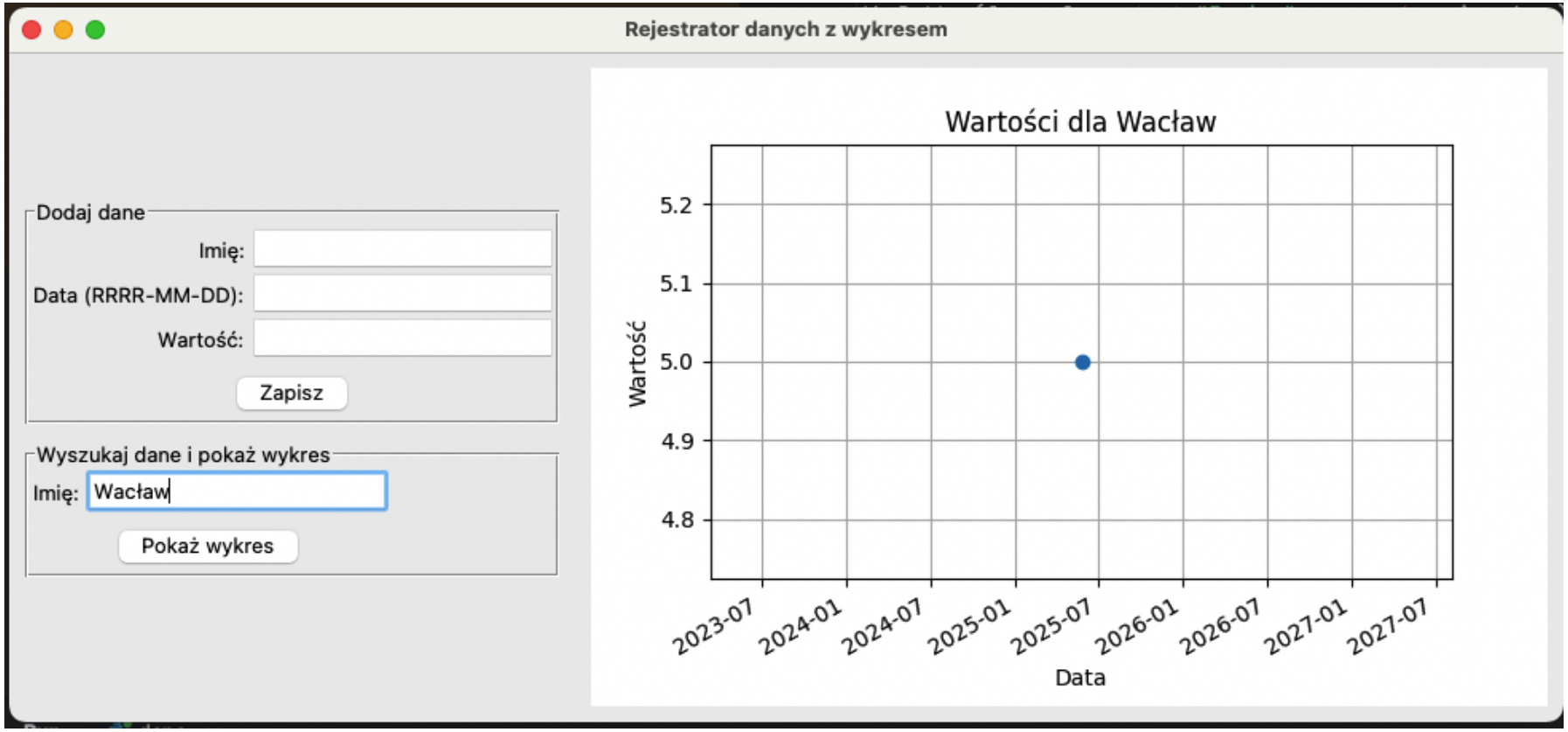
Istotnym problemem jest wyświetlanie wykresu w jednym oknie. Wymaga to kilku zmian w kodzie i zastosowania:

matplotlib.backends.backend_tkagg.FigureCanvasTkAgg do wyświetlania wykresu w tkinter.

Prezentacja danych w Pythonie

```
import tkinter as tk
from tkinter import messagebox
import pandas as pd
from datetime import datetime

import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg

PLIK_CSV = "dane.csv"

# --- Funkcje ---

def zapisz_dane():
    imie = entry_imie.get().strip()
    data = entry_data.get().strip()
    wartosc = entry_wartosc.get().strip()

    if not imie or not data or not wartosc:
        messagebox.showerror("Błąd", "Wszystkie pola muszą być wypełnione.")
        return

    try:
        data_parsed = datetime.strptime(data, "%Y-%m-%d")
        wartosc_float = float(wartosc)
    except:
        messagebox.showerror("Błąd", "Niepoprawny format daty (RRRR-MM-DD) lub liczby.")
        return

    nowy_wiersz = pd.DataFrame([[imie, data_parsed, wartosc_float]], columns=["Imię",
"Data", "Wartość"])

    try:
        istnieje = pd.read_csv(PLIK_CSV)
        nowy_wiersz.to_csv(PLIK_CSV, mode='a', index=False, header=False)
    except FileNotFoundError:
        nowy_wiersz.to_csv(PLIK_CSV, index=False)

    messagebox.showinfo("Sukces", "Dane zapisane!")
    entry_imie.delete(0, tk.END)
    entry_data.delete(0, tk.END)
    entry_wartosc.delete(0, tk.END)

def wyszukaj_i_wyswietl():
    imie_szukane = entry_szukaj.get().strip()

    try:
        df = pd.read_csv(PLIK_CSV, parse_dates=["Data"])
    except FileNotFoundError:
        messagebox.showerror("Błąd", "Brak pliku z danymi.")
        return

    df_osoba = df[df["Imię"].str.lower() == imie_szukane.lower()]

    if df_osoba.empty:
        messagebox.showinfo("Brak danych", f"Brak danych dla {imie_szukane}.")
        # Wyczyść wykres, jeśli jest
        clear_plot()
        return

    df_osoba = df_osoba.sort_values("Data")

    # Rysowanie wykresu
    fig.clear()
    ax = fig.add_subplot(111)
    ax.plot(df_osoba["Data"], df_osoba["Wartość"], marker='o')
    ax.set_title(f"Wartości dla {imie_szukane}")
    ax.set_xlabel("Data")
    ax.set_ylabel("Wartość")
    ax.grid(True)
    fig.autofmt_xdate()

    canvas.draw()

def clear_plot():
    fig.clear()
    canvas.draw()

# --- GUI ---

root = tk.Tk()
root.title("Rejestrator danych z wykresem")

# Ramka główna - podział na dwie kolumny
frame_left = tk.Frame(root)
frame_left.pack(side=tk.LEFT, padx=10, pady=10)

frame_right = tk.Frame(root)
frame_right.pack(side=tk.LEFT, padx=10, pady=10, fill=tk.BOTH, expand=True)

# --- Formularz dodawania danych ---
frame_form = tk.LabelFrame(frame_left, text="Dodaj dane")
frame_form.pack(fill="x", pady=5)

tk.Label(frame_form, text="Imię:").grid(row=0, column=0, sticky="e")
tk.Label(frame_form, text="Data (RRRR-MM-DD):").grid(row=1, column=0, sticky="e")
tk.Label(frame_form, text="Wartość:").grid(row=2, column=0, sticky="e")

entry_imie = tk.Entry(frame_form)
entry_data = tk.Entry(frame_form)
entry_wartosc = tk.Entry(frame_form)

entry_imie.grid(row=0, column=1)
entry_data.grid(row=1, column=1)
entry_wartosc.grid(row=2, column=1)

tk.Button(frame_form, text="Zapisz", command=zapisz_dane).grid(row=3, columnspan=2,
pady=5)

# --- Wyszukiwanie danych ---
frame_search = tk.LabelFrame(frame_left, text="Wyszukaj dane i pokaż wykres")
frame_search.pack(fill="x", pady=5)

tk.Label(frame_search, text="Imię:").grid(row=0, column=0, sticky="e")
entry_szukaj = tk.Entry(frame_search)
entry_szukaj.grid(row=0, column=1)

tk.Button(frame_search, text="Pokaż wykres", command=wyszukaj_i_wyswietl).grid(row=1,
columnspan=2, pady=5)

# --- Panel wykresu ---
fig, ax = plt.subplots(figsize=(6, 4))
canvas = FigureCanvasTkAgg(fig, master=frame_right)
canvas.get_tk_widget().pack(fill=tk.BOTH, expand=True)

root.mainloop()
```

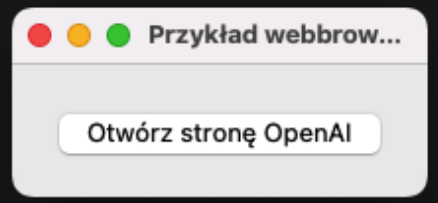
Prezentacja danych w Pythonie



Biblioteka Webbrowser

Stosowana jest do obsługi połączenia ze stroną internetową oraz wyświetlenia z niej danych.

```python
import tkinter as tk
import webbrowser

def otworz_strone():
    webbrowser.open("https://openai.com")

root = tk.Tk()
root.title("Przykład webbrowser")

btn = tk.Button(root, text="Otwórz stronę OpenAI", command=otworz_strone)
btn.pack(padx=20, pady=20)

root.mainloop()
```



Funkcja	Opis
open(url)	Otwiera URL w nowej karcie lub oknie
open_new(url)	Wymusza otwarcie w **nowym oknie**
open_new_tab(url)	Wymusza otwarcie w **nowej karcie**

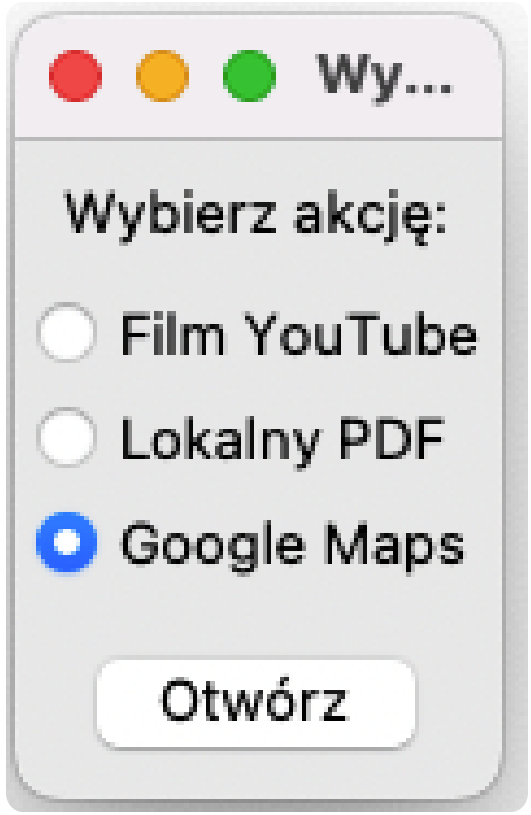
Biblioteka webbrowser

Po naciśnięciu przycisku otworzy się teledysk pewnego „wiecznie młodego” piosenkarza ...

```
import tkinter as tk
import webbrowser


def otworz_youtube():


webbrowser.open("https://www.youtube.com/watch?v
=dQw4w9WgXcQ")  # przykładowy link


root = tk.Tk()
root.title("Otwórz film")


btn = tk.Button(root, text="Otwórz film na YouTube",
command=otworz_youtube)
btn.pack(padx=20, pady=20)


root.mainloop()
```

Biblioteka Webbrowser

```
import tkinter as tk
import webbrowser
import os

def otworz_link():
    wybor = opcja.get()
    if wybor == "Film YouTube":
        webbrowser.open("https://www.youtube.com/watch?v=dQw4w9WgXcQ")
    elif wybor == "Lokalny PDF":
        pdf = os.path.abspath("dokument.pdf")
        webbrowser.open(f"file://{pdf}")
    elif wybor == "Google Maps":
        webbrowser.open("https://www.google.com/maps/place/Wrocław")

root = tk.Tk()
root.title("Wybierz co otworzyć")

opcja = tk.StringVar(value="Film YouTube")

tk.Label(root, text="Wybierz akcję:").pack(pady=5)

tk.Radiobutton(root, text="Film YouTube", variable=opcja, value="Film
YouTube").pack(anchor="w")
tk.Radiobutton(root, text="Lokalny PDF", variable=opcja, value="Lokalny
PDF").pack(anchor="w")
tk.Radiobutton(root, text="Google Maps", variable=opcja, value="Google
Maps").pack(anchor="w")

tk.Button(root, text="Otwórz", command=otworz_link).pack(pady=10)

root.mainloop()
```



Biblioteka Webbrowser

Inne zastosowania:

wyszukiwanie

```
import webbrowser

zapytanie = "Jak używać tkinter"
webbrowser.open(f"https://www.google.com/search?q={zapytanie.replace(' ', '+')}")
```

poczta

```
webbrowser.open("mailto:nauczyciel@szkola.pl?subject=Temat&body=Wiadomość")
```

pliki lokalne

```
import os
import webbrowser

plik_html = os.path.abspath("raport.html")
plik_pdf = os.path.abspath("faktura.pdf")
webbrowser.open(f"file://{plik_html}")
webbrowser.open(f"file://{plik_pdf}")
```

Biblioteka Webbrowser

```
start = "Wrocław"
cel = "Kraków"
link = f"https://www.google.com/maps/dir/{start.replace(' ', '+')}/{cel.replace(' ', '+')}/"
webbrowser.open(link)
```

Prosty sposób na wyszukanie trasy w mapach Google.

Przed Wami projekt wykorzystania tej biblioteki zawierający przejrzysty interfejs użytkownika ze wszystkimi sposobami wykorzystania webbrowser. Można zastosować ikony załączone w plikach lub charcode.

Biblioteka Webbrowser

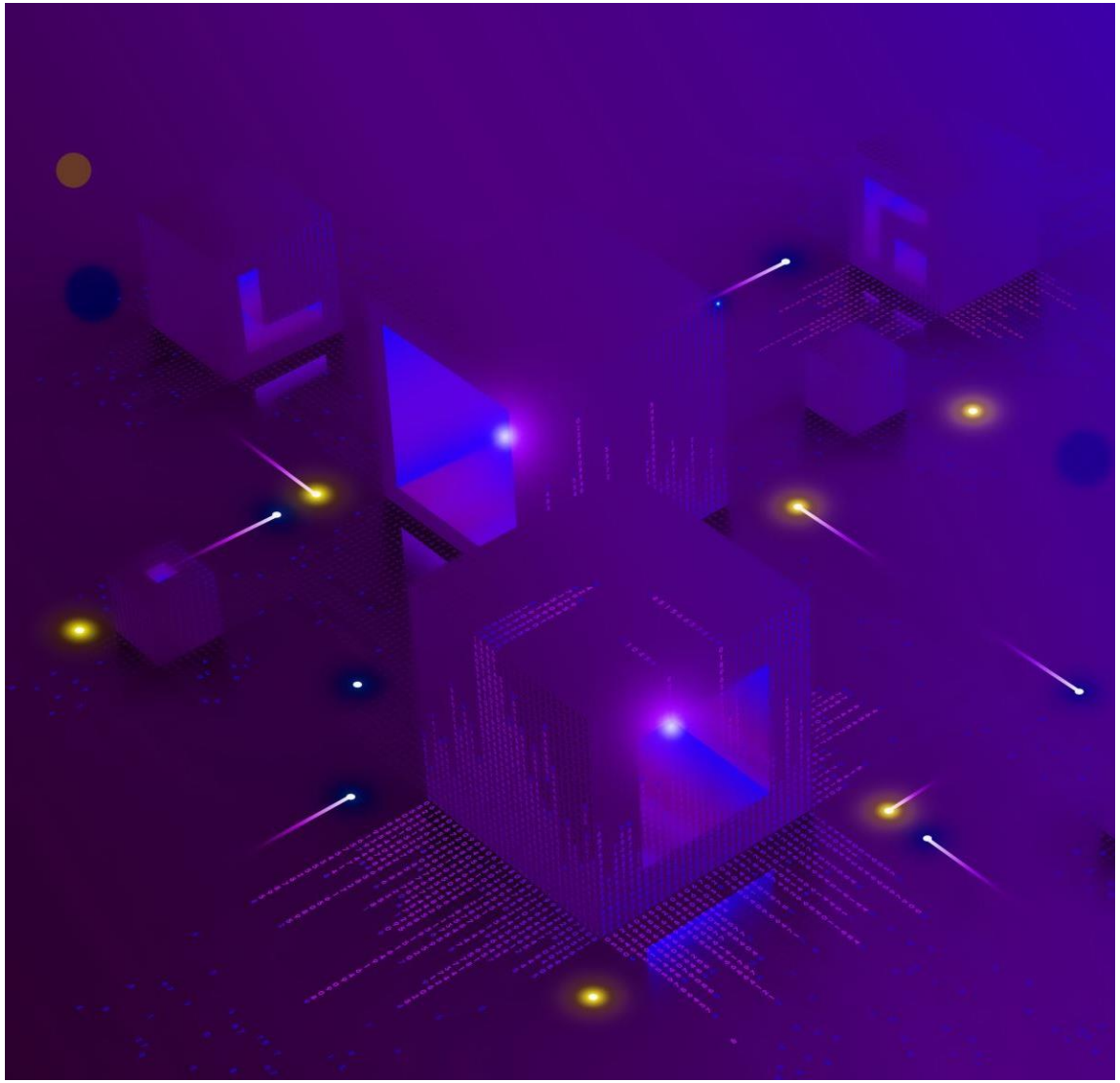
```
import tkinter as tk
import webbrowser
import os

# --- Funkcje otwierające ---
def otworz_dokumentacja():
    webbrowser.open("https://docs.python.org/3/")

def otworz_google():
    zapytanie = "jak używać tkinter"

webbrowser.open(f"https://www.google.com/search?q={zapy
tanie.replace(' ', '+')}")

def otworz_formularz():
    webbrowser.open("https://forms.gle/example")  # ← zmień
na własny link

def otworz_pdf():
    sciezka = os.path.abspath("dokument.pdf")
    webbrowser.open(f"file://{sciezka}")

def otworz_mape():
    start = "Wrocław"
    cel = "Kraków"
    url = f"https://www.google.com/maps/dir/{start}/{cel}"
    webbrowser.open(url)

def otworz_youtube():

webbrowser.open("https://www.youtube.com/results?search
_query=python+tutorial")

def wyslij_email():

webbrowser.open("mailto:test@example.com?subject=Zapyt
anie&body=Proszę o kontakt")

# --- GUI ---
root = tk.Tk()
root.title("Otwieranie linków z ikonami Unicode")
root.geometry("430x450")
root.configure(bg="#f0f0f0")

# --- Lista przycisków z Unicode zamiast ikon ---
przyciski = [
    ("🐍 Python Dokumentacja", otworz_dokumentacja),
    ("🔍 Wyszukaj w Google", otworz_google),
    ("📝 Formularz Google", otworz_formularz),
    ("📄 Otwórz PDF", otworz_pdf),
    ("🗺️ Mapa: Wrocław → Kraków", otworz_mape),
    ("📺 YouTube: Python", otworz_youtube),
    ("✉️ Wyślij e-mail", wyslij_email),
]

# --- Tworzenie przycisków ---
for tekst, funkcja in przyciski:
    btn = tk.Button(root, text=tekst, padx=10, pady=5,
width=40,
                    anchor="w", font=("Segoe UI Emoji", 11),
                    command=funkcja, bg="white", relief="raised")
    btn.pack(pady=7)

root.mainloop()
```

Biblioteka Webbrowser

Estetyczny wygląd, ikony i wiele możliwości w jednym programie, który można wykorzystać jako element kolejnego programu. Idealny jako element menu.

Podsumowanie - pytania

1. Do czego służy mapping w Tkinter?
2. Do czego służy TreeView?
3. Do czego służy MySQL.Connector?
4. Co jest wymagane do połączenia pliku z layout z plikiem wykonywalnym?
5. Czy plik UI można podłączyć do projektu?
6. Do czego służy Matplotlib i Pyplot?
7. Czy przez bibliotekę Webbrowser można wysłać e-mail?
8. Czy można zastosować Bootstrap w języku Python?

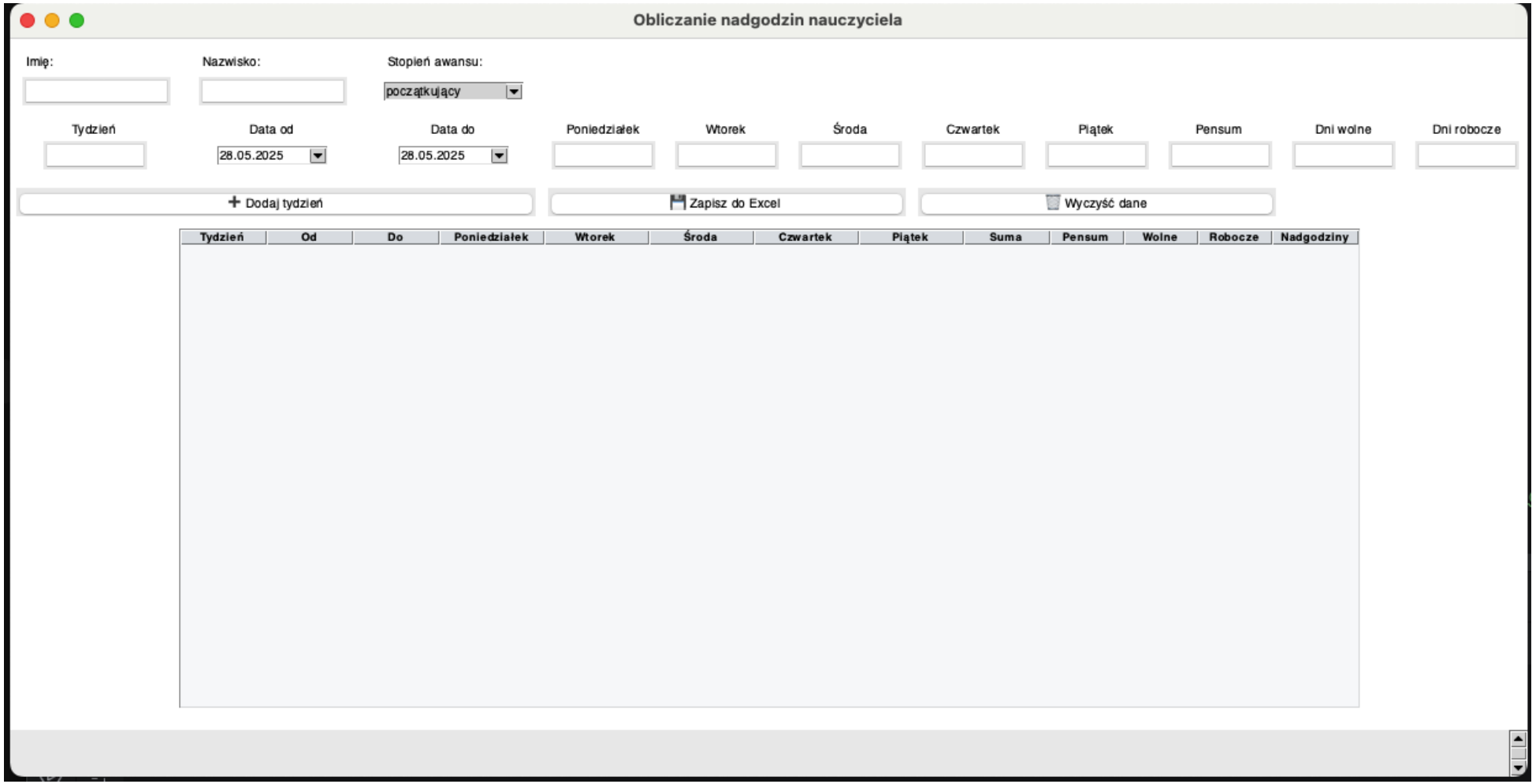


Zadania do samodzielnego wykonania

Zadanie 1

Aplikacja oblicza nadgodziny pracy nauczyciela według cyklu tygodniowego. Dane zapisują się jednocześnie w tabeli oraz pliku arkusza kalkulacyjnego.

Do wykonania zadania potrzebne będą biblioteki openpyxl oraz tkcalendar.



Zadania do samodzielnego wykonania

Jak obliczyć nadgodziny? Służy do tego wzór:

L=G−P×(1−nt)L=G−P×(1−tn)

gdzie:

•LL – liczba nadgodzin w tygodniu,
•GG – liczba godzin zrealizowanych w danym tygodniu,
•PP – tygodniowe pensum nauczyciela (np. 18 godzin),
•nn – liczba dni wolnych od pracy (np. święta, urlop, zwolnienie),
•tt – liczba dni roboczych w tygodniu (zwykle 5).

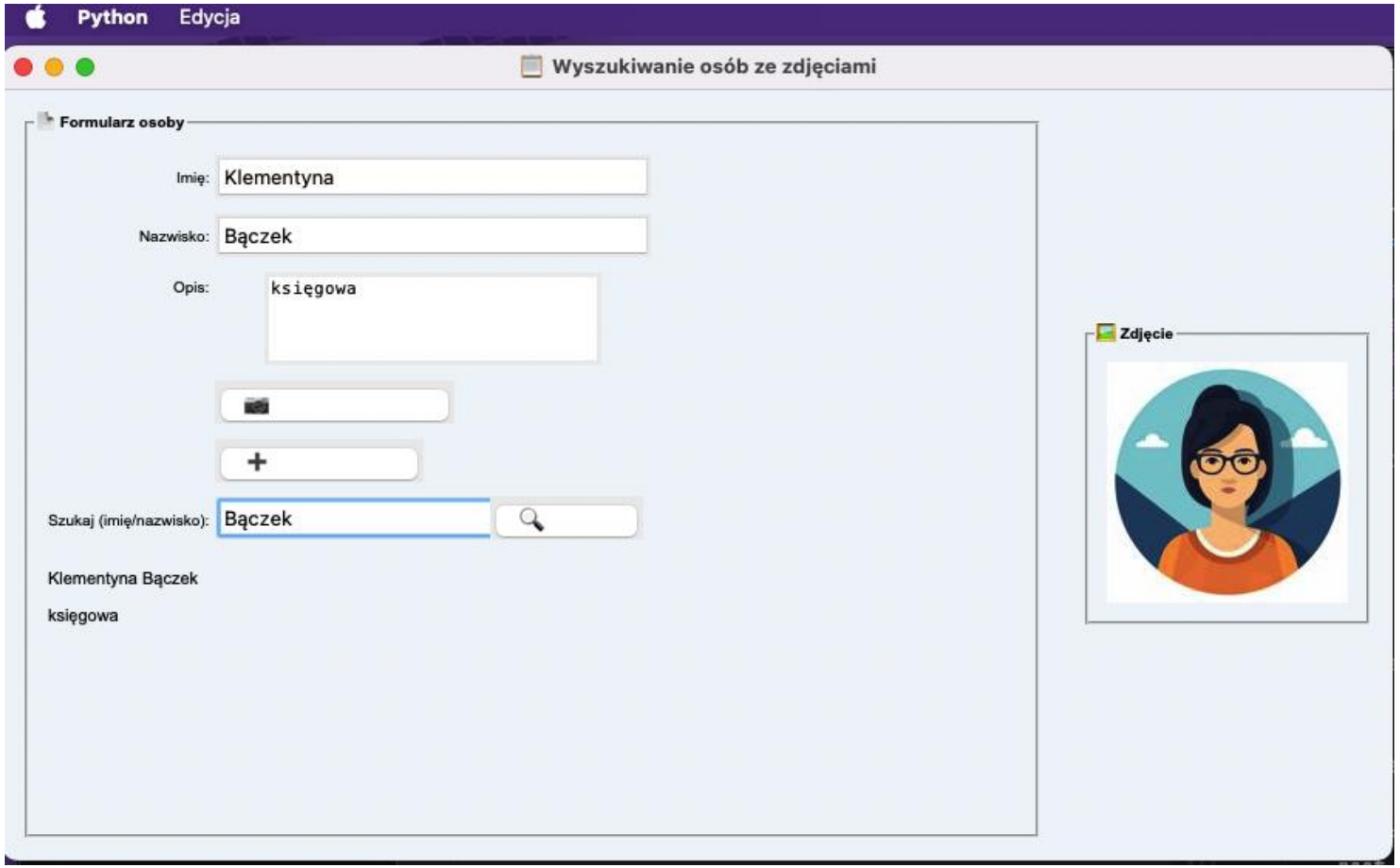
Przykład obliczenia:

Załóżmy:

•Nauczyciel przepracował w tygodniu 22 godziny: G=22G=22,
•Jego pensum to 18 godzin: P=18P=18,
•W tygodniu był 1 dzień wolny: n=1n=1,
•Tydzień roboczy miał 5 dni: t=5t=5
L=22−18×(1−15)=22−18×45=22−14.4=7.6L=22−18×(1−51)=22−18×54=22−14.4=7.6

Wynik: 7,6 nadgodzin w tym tygodniu.

Zadania do samodzielnego wykonania



Zadanie 2

Program służy do gromadzenia danych o pracownikach. Pozwala dodać zdjęcia z dysku do struktury projektu oraz je wyświetlić dzięki wyszukiwarce.

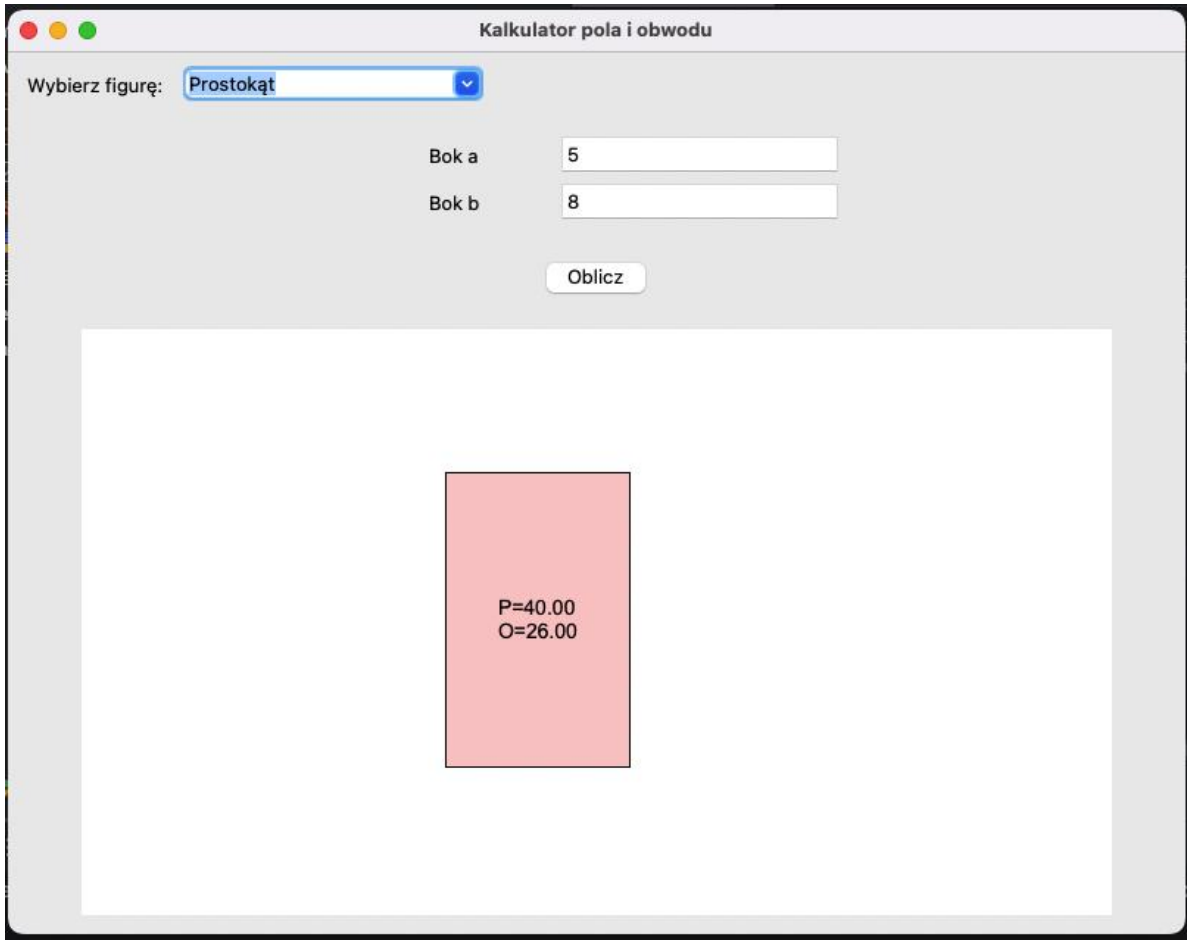
W górnym menu „edycja” znajdują się opcje „wytnij”, „kopiuj” i „wklej”.

Dane zapisują się w pliku JSON.

Zadania do samodzielnego wykonania

Zadanie 3 (na 5)

Napisz program liczący pola i obwody różnych figur. Użytkownik może wybrać prostokąt, kwadrat, koło i trójkąt równoboczny a następnie wpisuje dane w pojawiające się pola. Po obliczeniu program rysuje figurę a obliczenia wpisuje wewnątrz narysowanej figury.

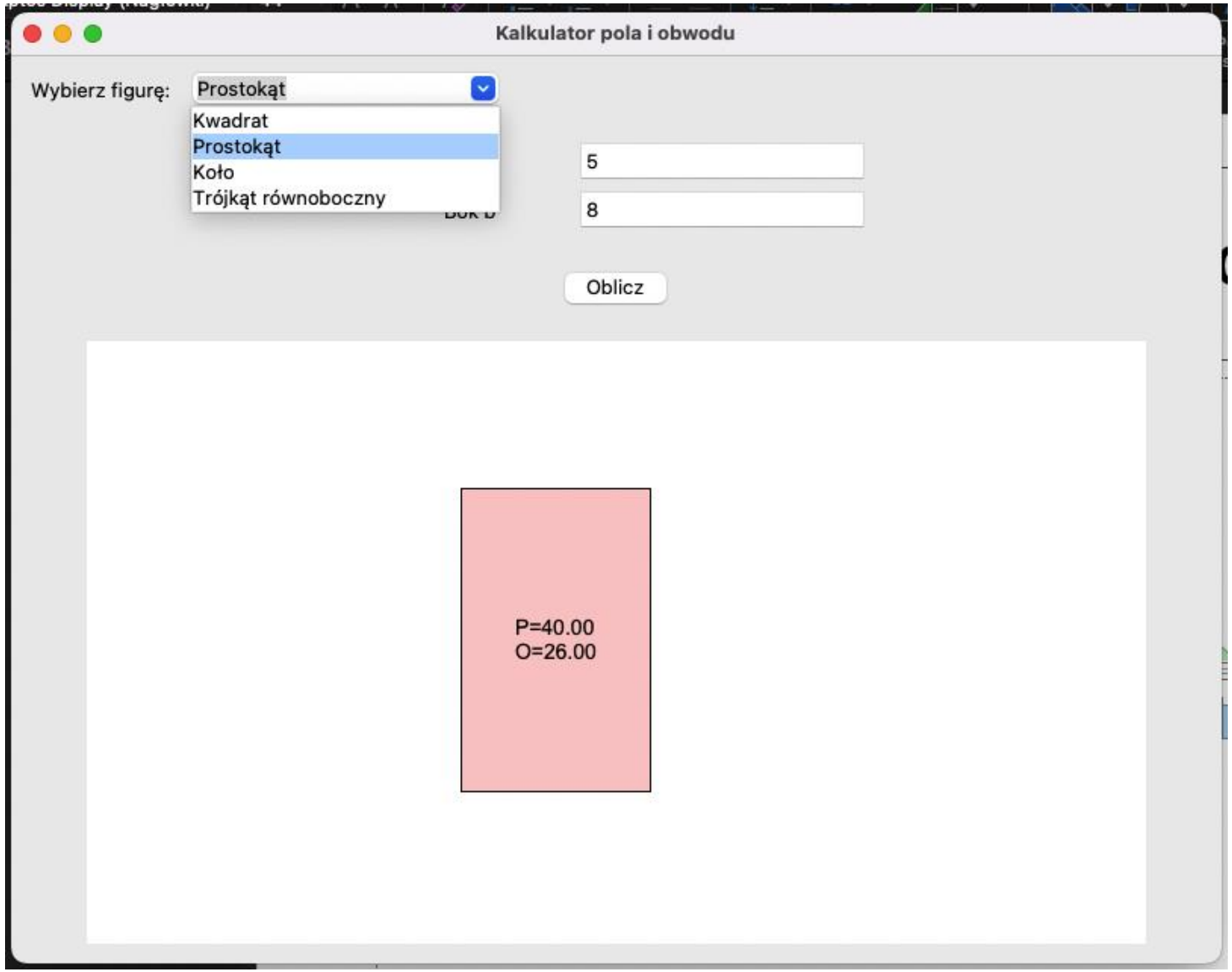


Zadania do samodzielnego wykonania

Pojawienie się pól do wpisania danych jest zależne od wyboru figury.

Do wyrysowania figur użyto Canvas.

Na wyższą ocenę dodać więcej figur: trapez, romb, lub inne.

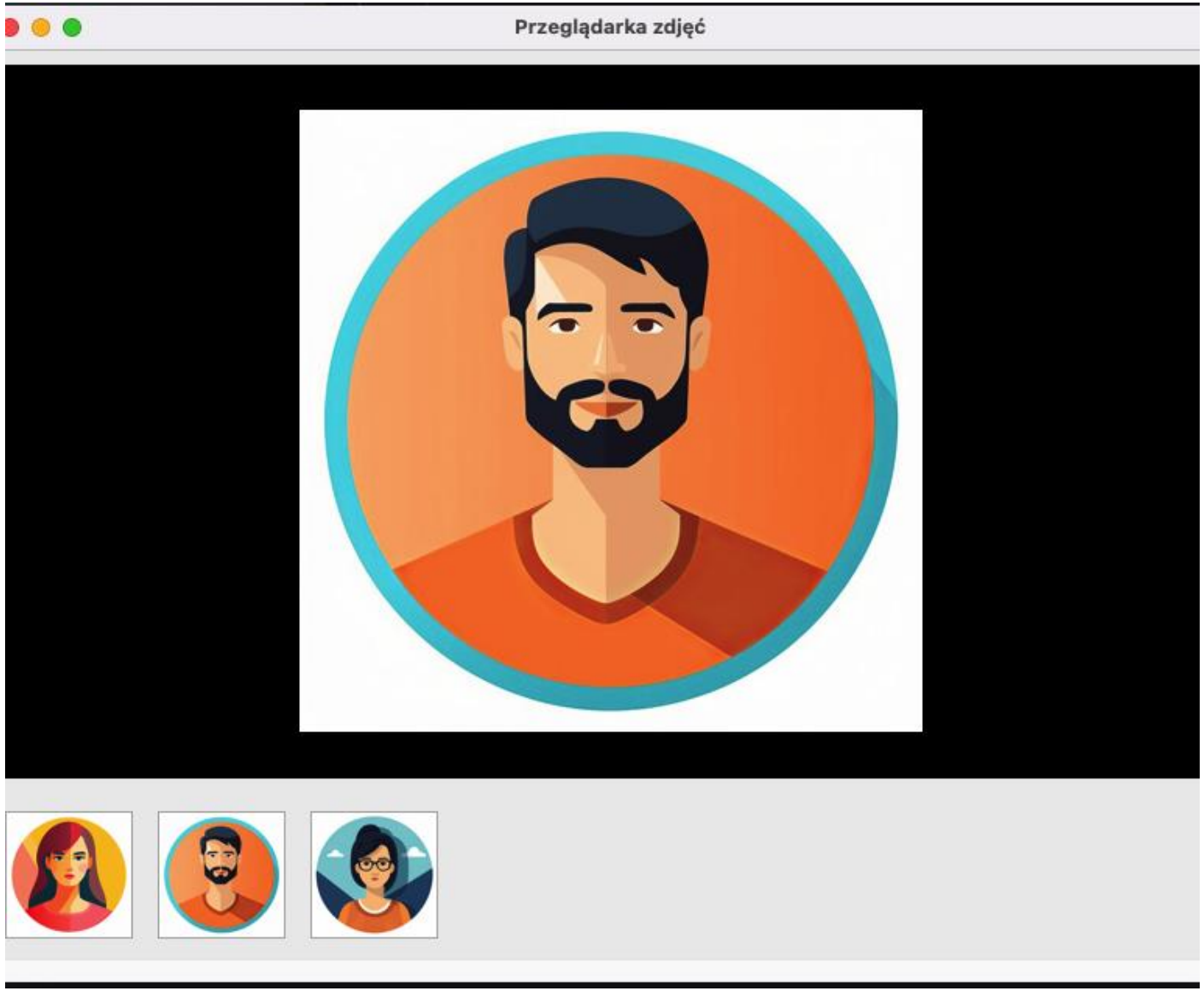


Zadania do samodzielnego wykonania

Zadanie 4

Zadanie w formie przeglądarki zdjęć. W dolnej części okna znajdują się miniatury.
Po naciśnięciu miniatury obraz w pełnej skali wyświetla się w górnej części okna.

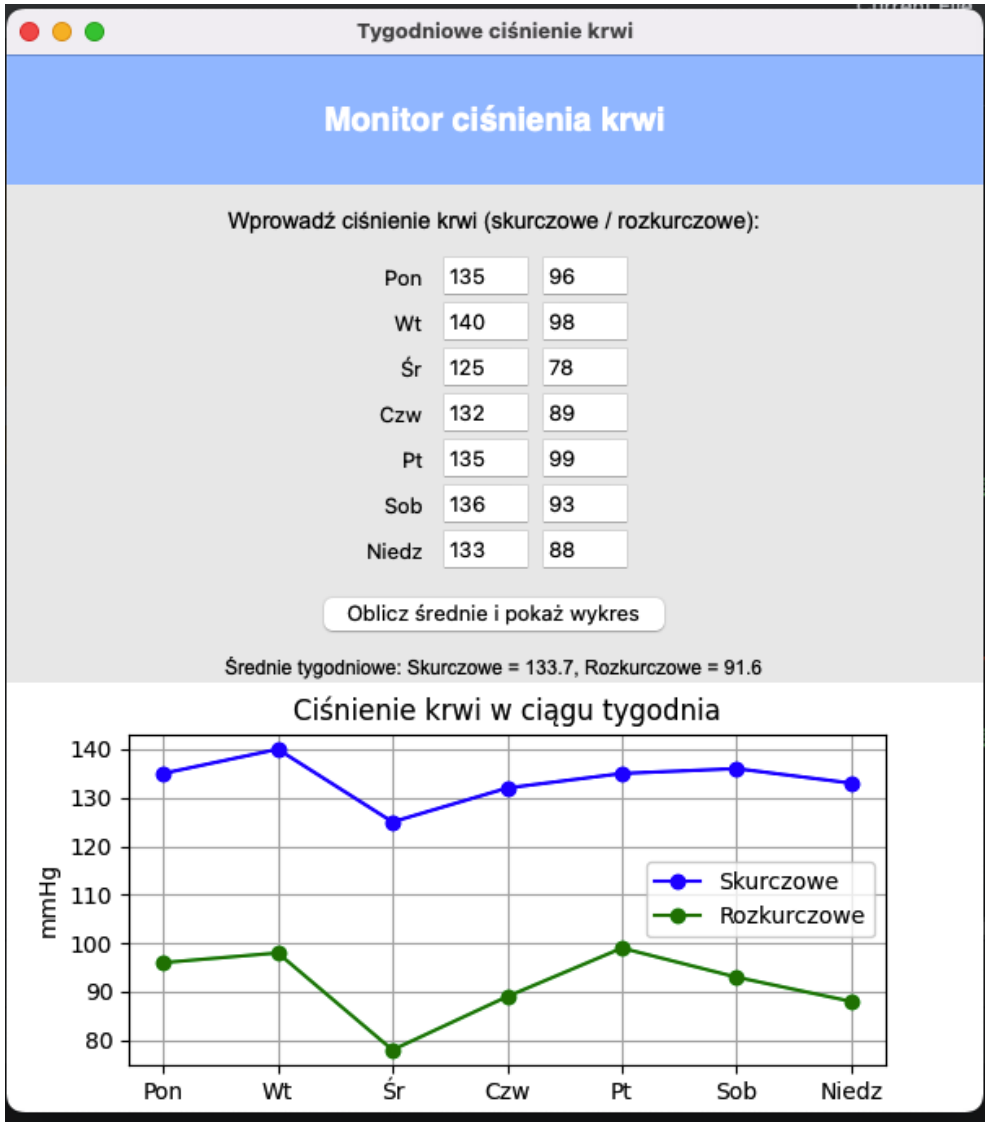
Rozmiar miniatury wynosi 20% rozmiaru całego obrazka.
Wykorzystano Tk Image, OS oraz Canvas.



Zadania do samodzielnego wykonania

Zadanie 5 (zadanie na 5)

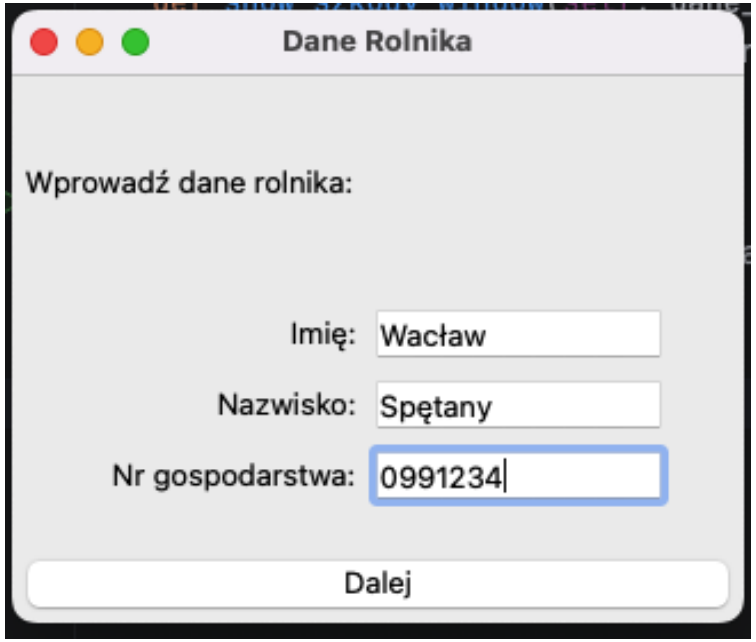
Program pobiera od użytkownika wartości ciśnień. Wylicza z nich średnią tygodniową a następnie przenosi dane na wykres. Posiada estetyczny nagłówek, do którego użyto formatowania czcionek, Canvas oraz Pack jako sposób na rozmieszczenie elementów.

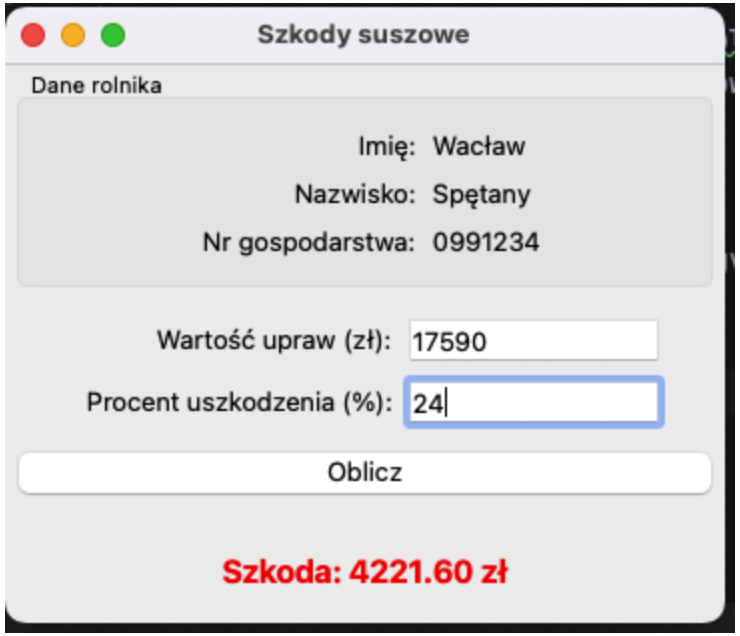


Zadania do samodzielnego wykonania

Zadanie 6

Twoim zadaniem jest wykonanie programu do obliczania szkód w rolnictwie. Interfejs użytkownika został utworzony za pomocą qT Designer i zaimportowany do projektu.



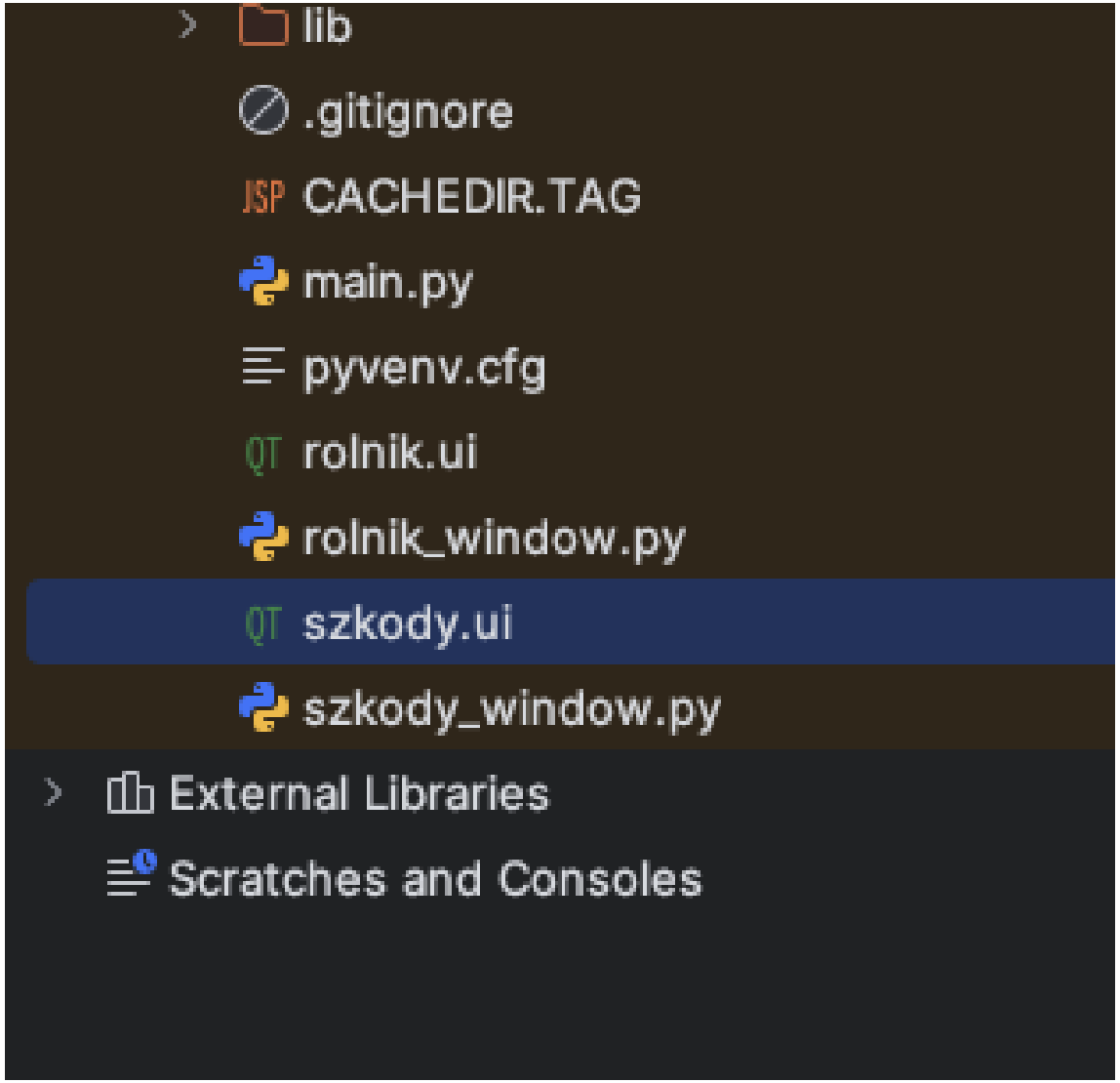


Zadania do samodzielnego wykonania

Przed Tobą struktura projektu:

- Pliki PY zawierają logikę programu,
- UI zawierają wygląd opisany za pomocą XML,
- Program uruchamia Main.

Chcesz dodatkową ocenę? Dodaj do projektu estetyczny wygląd zawierający formatowanie tekstu, pól tekstowych, inne położenie elementów oraz tło w postaci obrazka.

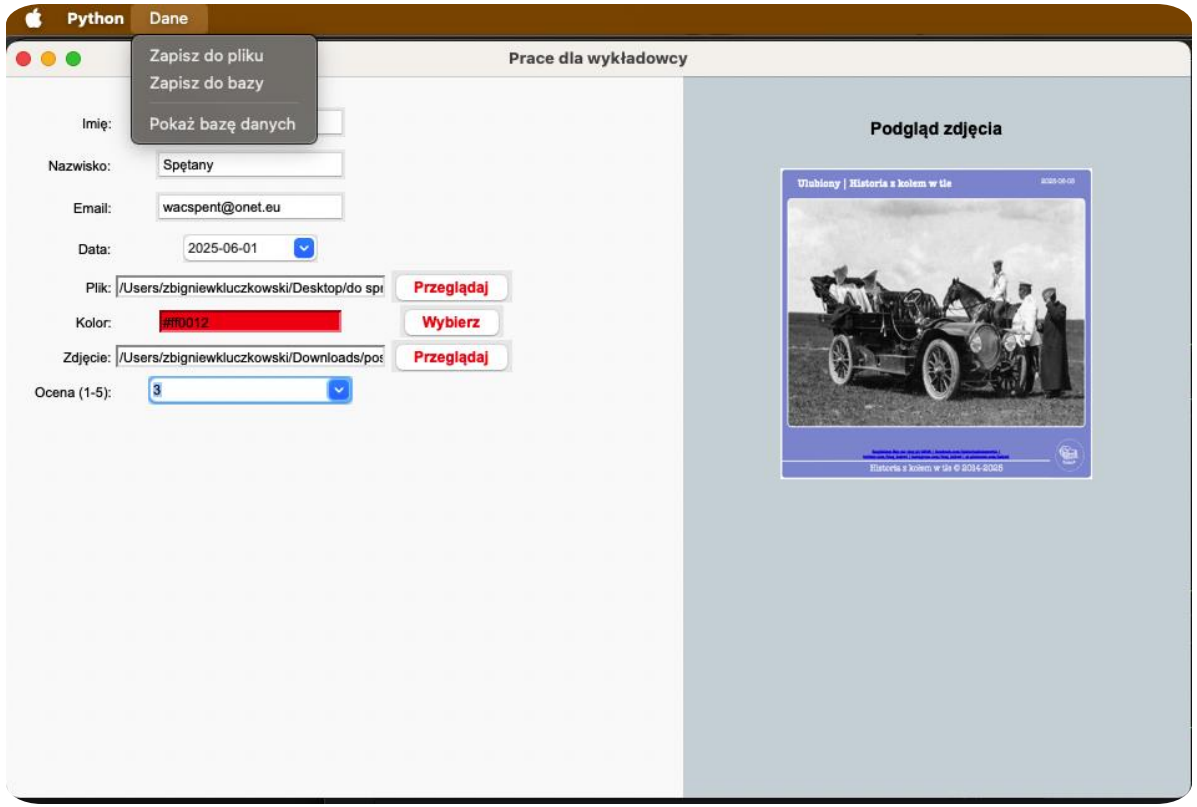


Zadania do samodzielnego wykonania

Zadanie 7

Aplikacja przesyła do pliku. Student wypełnia formularz załączając do niego plik z pracą. Im ciemniejszy kolor wybierze tym oczekuje szybszego sprawdzenia pracy. Zdjęcie pobrane przez użytkownika zostaje wyświetlone w drugiej kolumnie.

Aplikacja posiada estetyczne tło, podział na dwie kolumny oraz responsywny układ elementów.



Zadania do samodzielnego wykonania

Zadanie 8 (na ocenę 5)

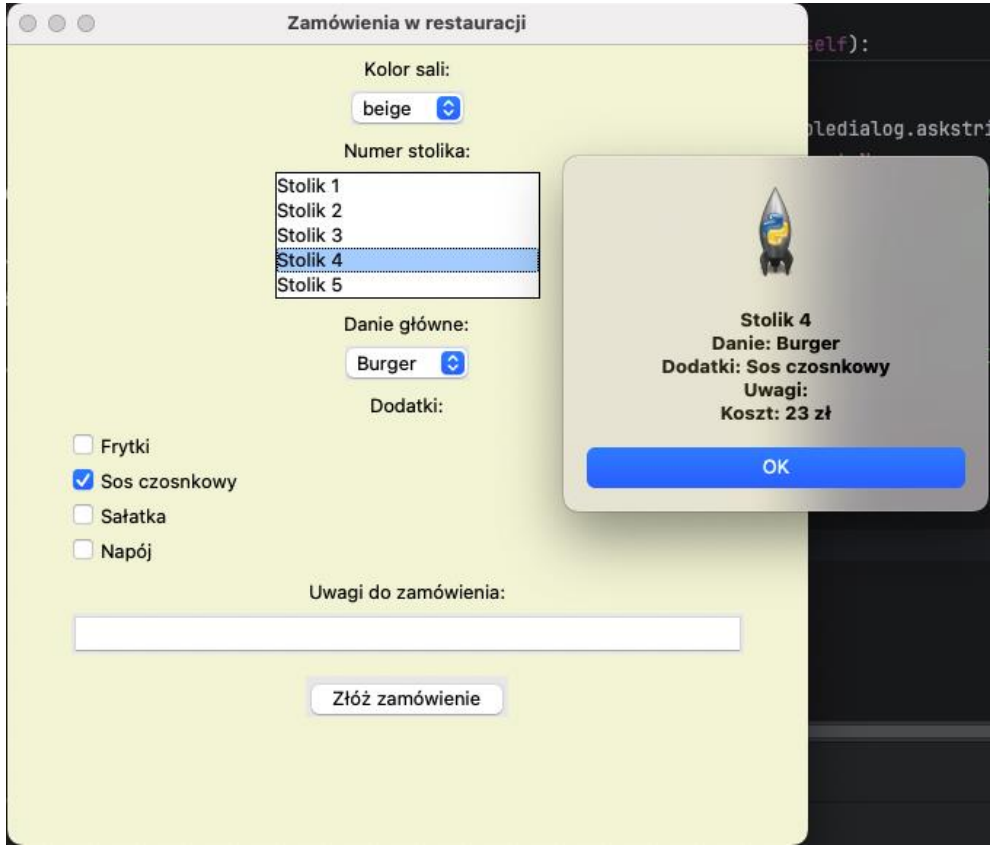
Stanowi rozwinięcie zadania 7. Zawiera dodatkowe formatowania oraz możliwość zapisu danych do bazy danych SQLite.

Zamiast przycisków funkcyjnych zastosowano rozwijane menu i dodano ocenę proponowaną przez studenta.

Zadania do samodzielnego wykonania

Zadanie 9

Kierowcy wpisują dane na temat spalania ich pojazdów. Jeżeli wybierze spalanie większe niż 10 litrów to przekroczy normę i na etykiecie wyświetli się kolor czerwony. Jeżeli nie to wyświetli się kolor zielony. Dane są zapisywane w liście pod przyciskiem oraz w bazie SQLite.



Zadania do samodzielnego wykonania

Zadanie 10

Program do obsługi restauracji, za pomocą którego można składać zamówienia.

Kelner wybiera jedną z 5 sal opisanych kolorami, w każdej z nich jest 5 stolików.

Klient może wybrać kilka dań, frytki, sos, sałatkę lub napój. Może dodać uwagi do zamówienia.

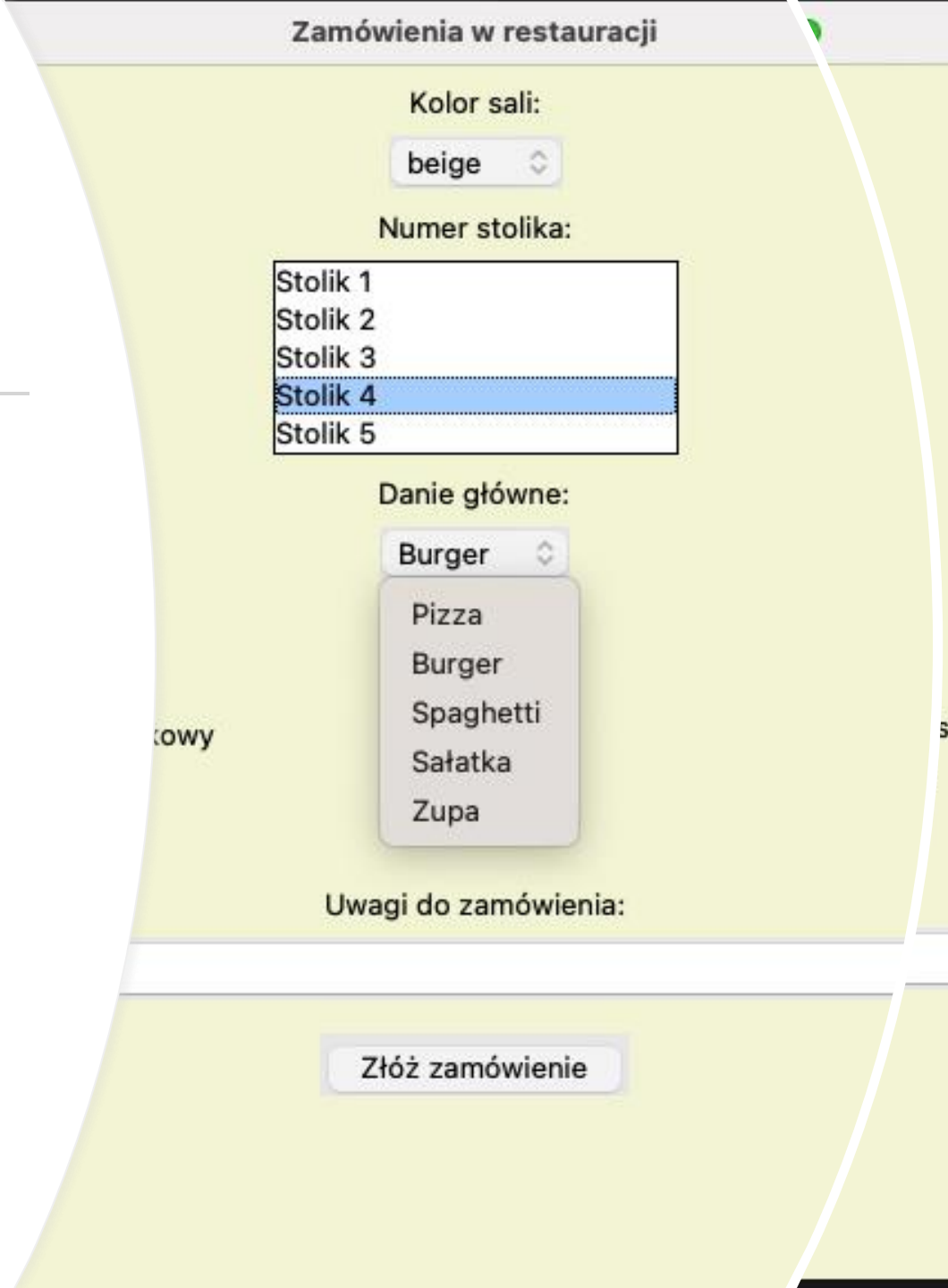
Aplikacja oblicza koszt pojedynczego zamówienia.

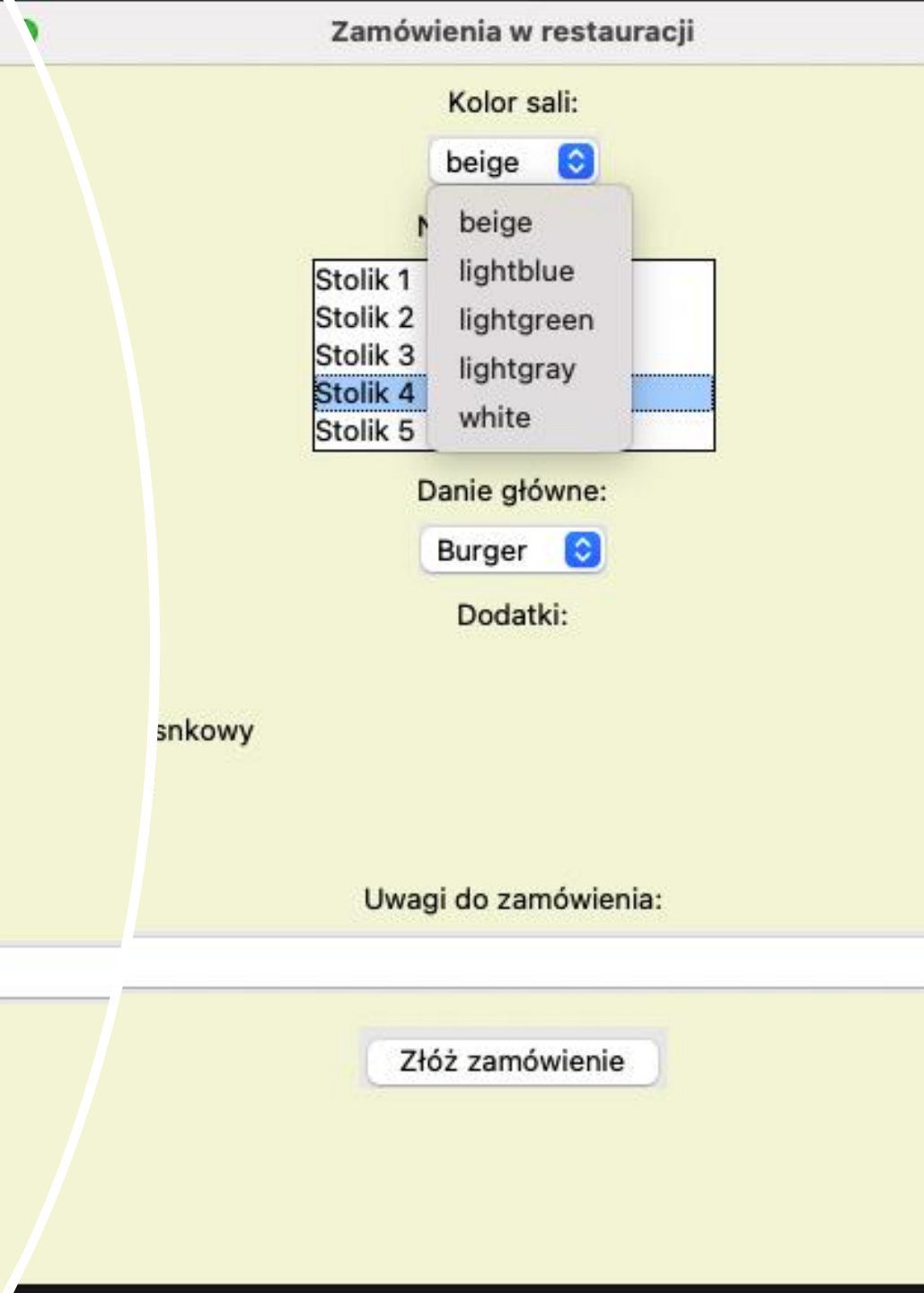
Zadania do samodzielnego wykonania

Dane są gromadzone w bazie danych SQLite.

Do projektu dodano estetyczne tło.

Ceny można dowolnie ustalić.





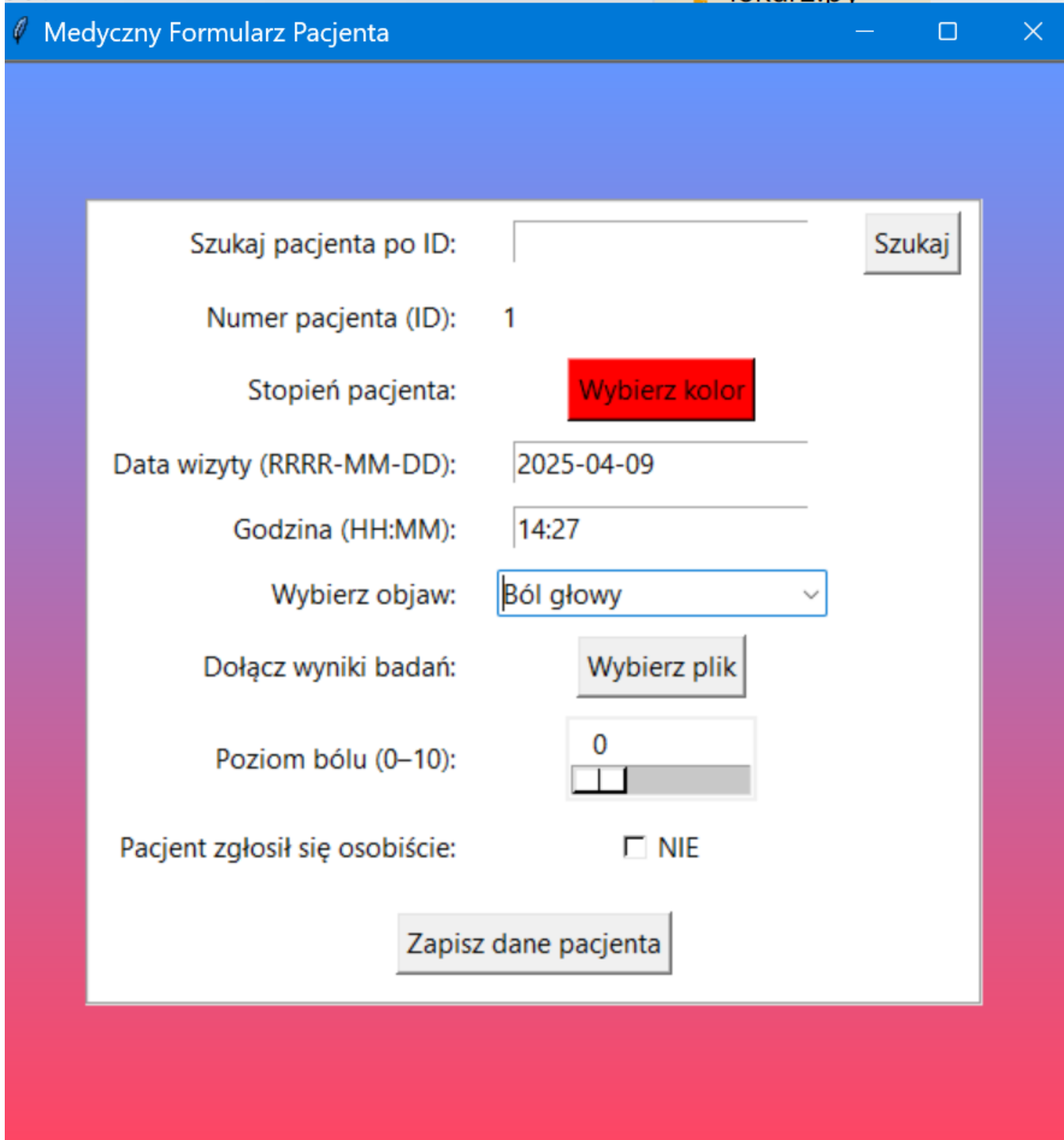
Zadania do samodzielnego wykonania

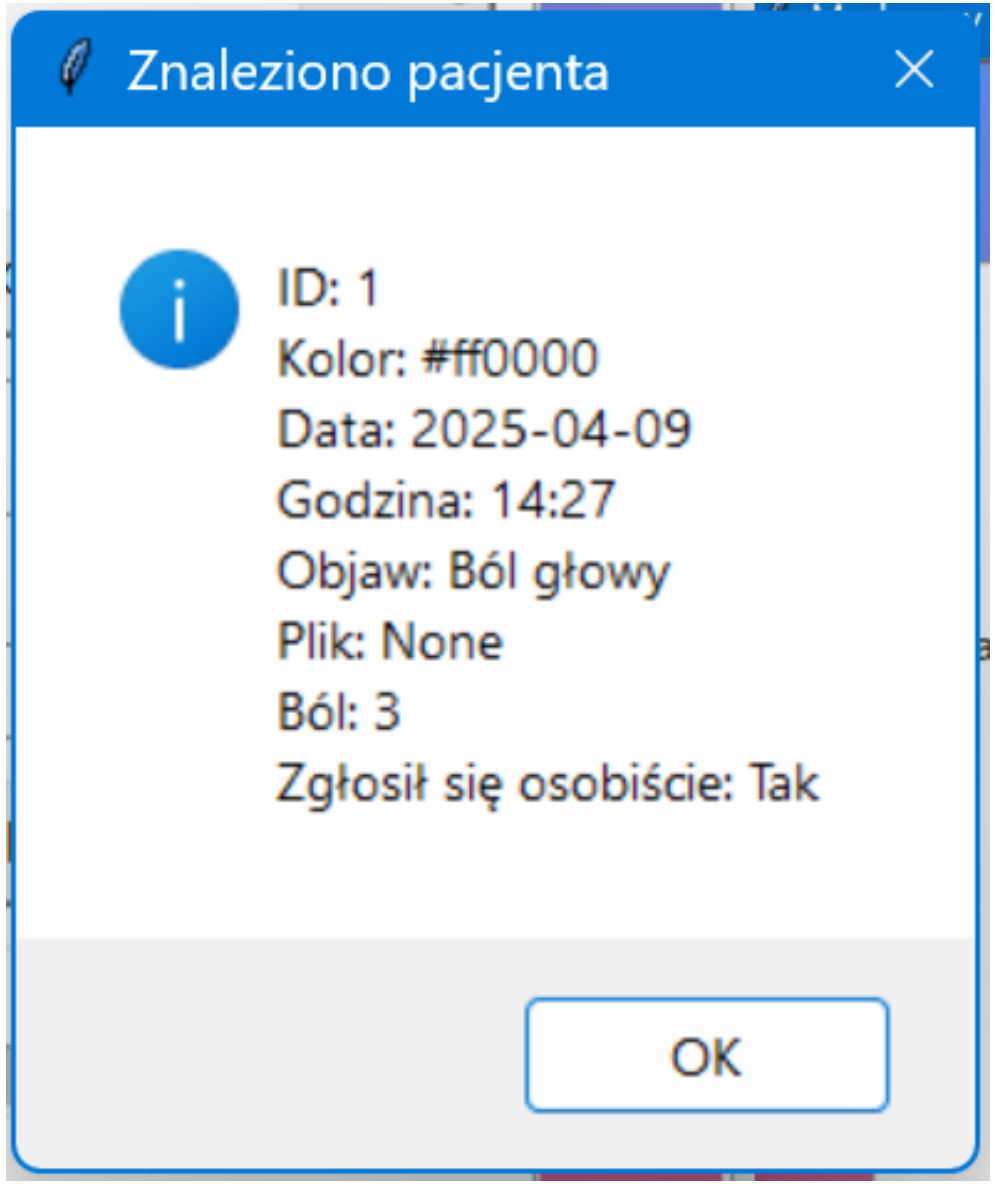
Zadanie 11

Celem tego zadania jest wykonanie aplikacji dla potrzeb Izby Przyjęć w szpitalu.

Dane są zapisywane z pliku JSON. Użytkownik może wyszukiwać dane po numerze ID.

Projekt zawiera pola: czas, plik, suwak oraz listę rozwijaną.





Zadania do samodzielnego wykonania

Lista rozwijana zawiera 5 typowych dolegliwości. Ocena bólu w skali od 0 do 10. Po wyborze koloru przycisk zmienia kolor na wybrany.

Po zaznaczeniu pola checkbox etykieta z pomyślnego „NIE” zmienia się na „TAK”.

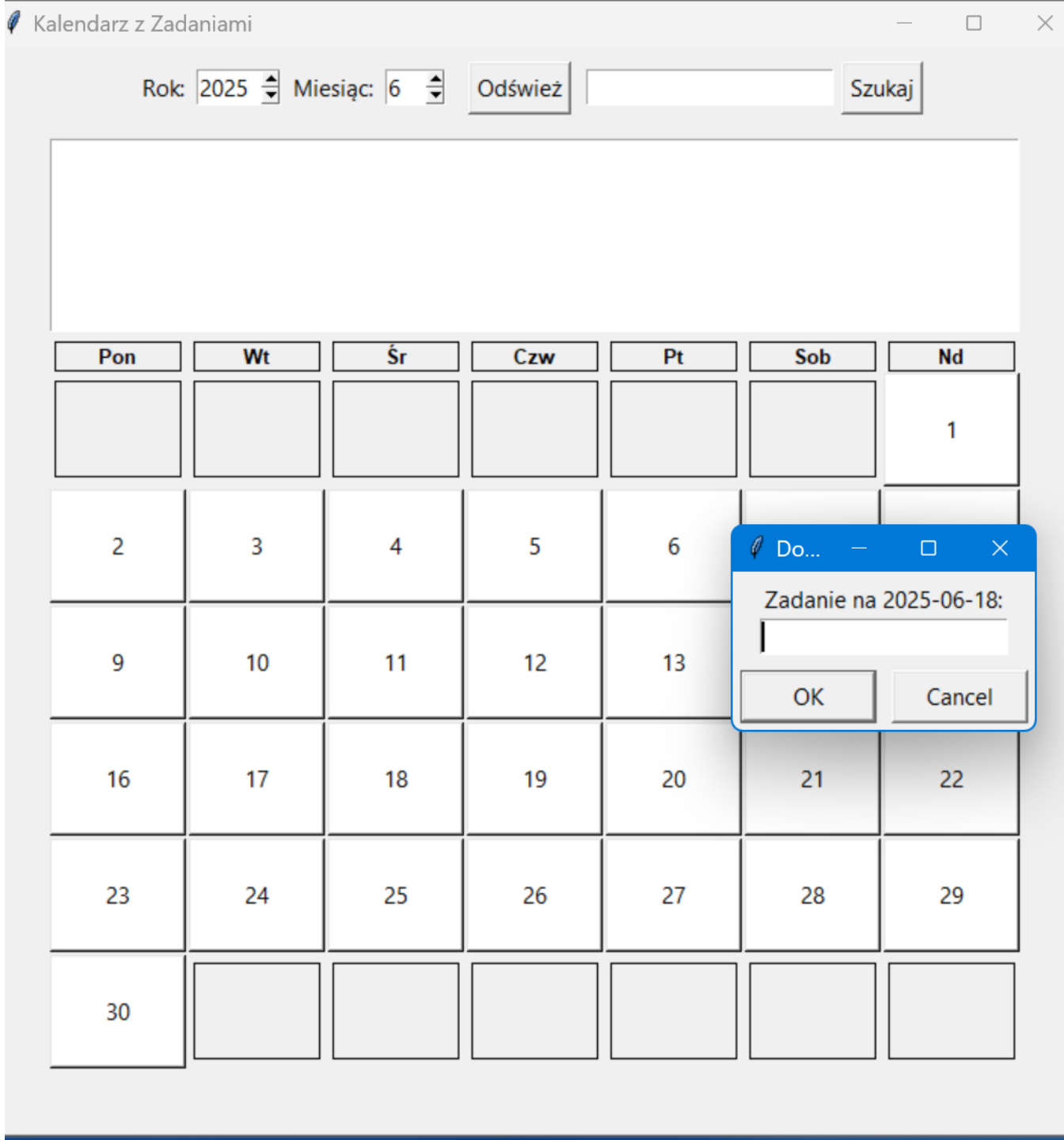
Po wyszukaniu wyświetla się okno z wpisanymi danymi.

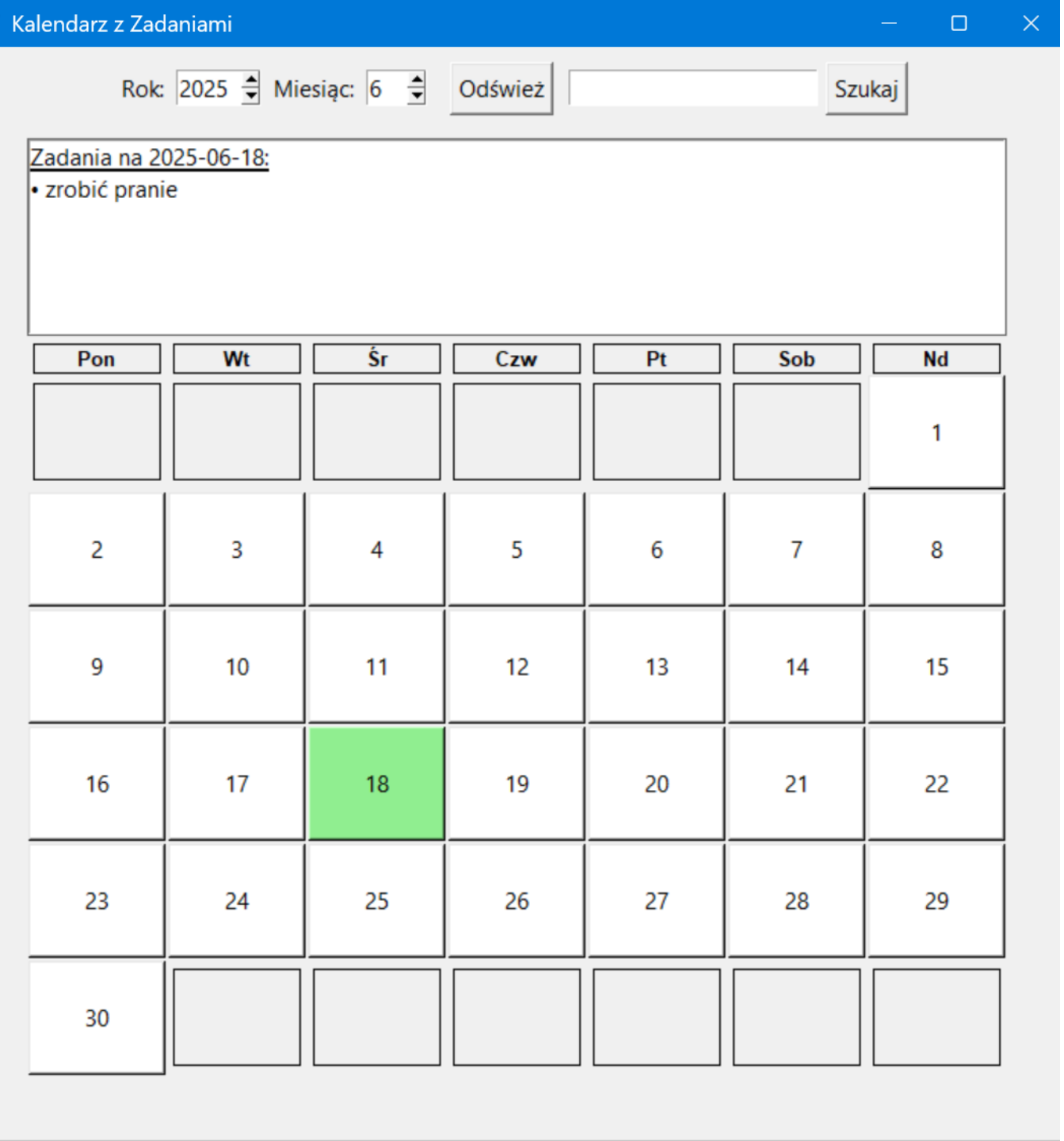
Zadania do samodzielnego wykonania

Zadanie 12 (zadanie na 5)

Zadanie w formie kalendarza. Użytkownik wprowadza treść zadania po kliknięciu w odpowiedni dzień miesiąca.

Wyszukiwarka pozwala wyszukać czynność i wyświetla ją w polu poniżej.





Zadania do samodzielnego wykonania

Po zapisaniu zadania na wybrany kolor podświetla się przycisk z dniem miesiąca.

Dane są zapisywane w pliku JSON i są pobierane przez wyszukiwarkę.

Projekt można rozwinąć dodając MENU z opcją wydrukowania danych za pomocą zrzutu ekranu.

Gry w okienkowym Pythonie

Konieczne i często używane biblioteki do gier w Tkinter:

- **tkinter**
  - Podstawowa biblioteka GUI do tworzenia okien, przycisków, canvasów itd.
  - Dostarcza narzędzia do rysowania grafiki i obsługi zdarzeń.
- **random**
  - Do losowania np. pozycji obiektów, ruchów przeciwników czy wyboru losowych elementów.
- **time** (opcjonalnie)
  - Do kontrolowania czasu, opóźnień lub pomiaru czasu reakcji.
- **math** (opcjonalnie)
  - Do obliczeń geometrycznych, ruchów, kolizji.

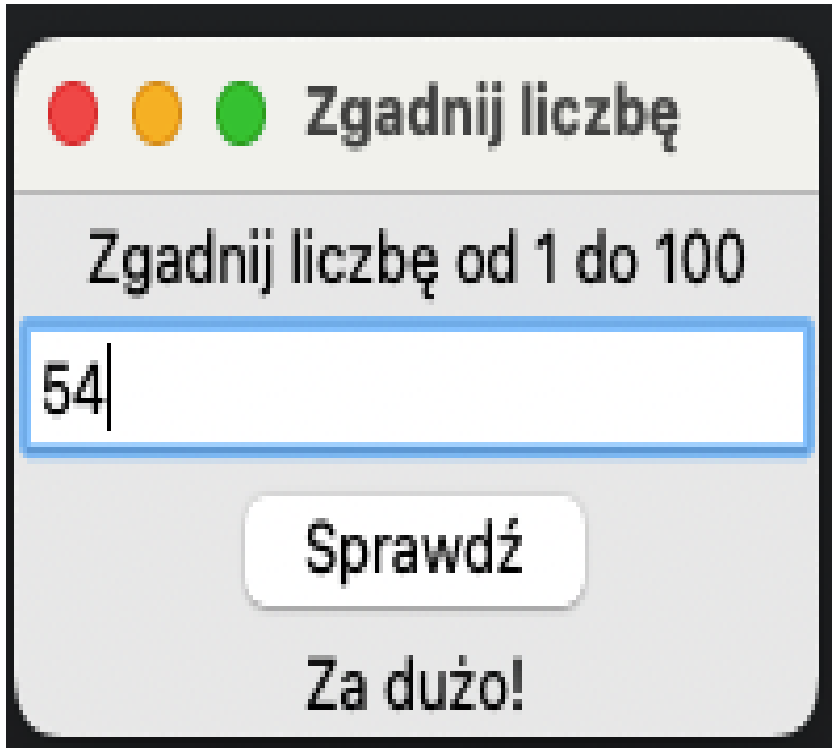
Przydatne dodatkowe biblioteki (niekonieczne, ale pomocne):

- **PIL / Pillow**
  - Do obsługi i wyświetlania obrazków, sprite'ów w grach.
- **pygame** (poza Tkinterem)
  - Bardziej zaawansowana biblioteka do tworzenia gier 2D, ale jeśli chcesz proste GUI, Tkinter wystarczy.

Gry w okienkowym Pythonie

Gra „Zgadnij liczbę”

Użytkownik zgaduje liczbę z przedziału.
Program w odpowiedzi podaje czy to za dużo czy za mało.




```
import tkinter as tk
import random

def sprawdz():
    try:
        guess = int(entry.get())
    except ValueError:
        label_result.config(text="Wpisz poprawną liczbę!")
        return

    if guess < number:
        label_result.config(text="Za mało!")
    elif guess > number:
        label_result.config(text="Za dużo!")
    else:
        label_result.config(text="Brawo! Trafiłeś!")
        btn_check.config(state=tk.DISABLED)

number = random.randint(1, 100)

root = tk.Tk()
root.title("Zgadnij liczbę")

tk.Label(root, text="Zgadnij liczbę od 1 do 100").pack()
entry = tk.Entry(root)
entry.pack()
btn_check = tk.Button(root, text="Sprawdź",
command=sprawdz)
btn_check.pack()
label_result = tk.Label(root, text="")
label_result.pack()

root.mainloop()
```

Ruch postaci w PyGame

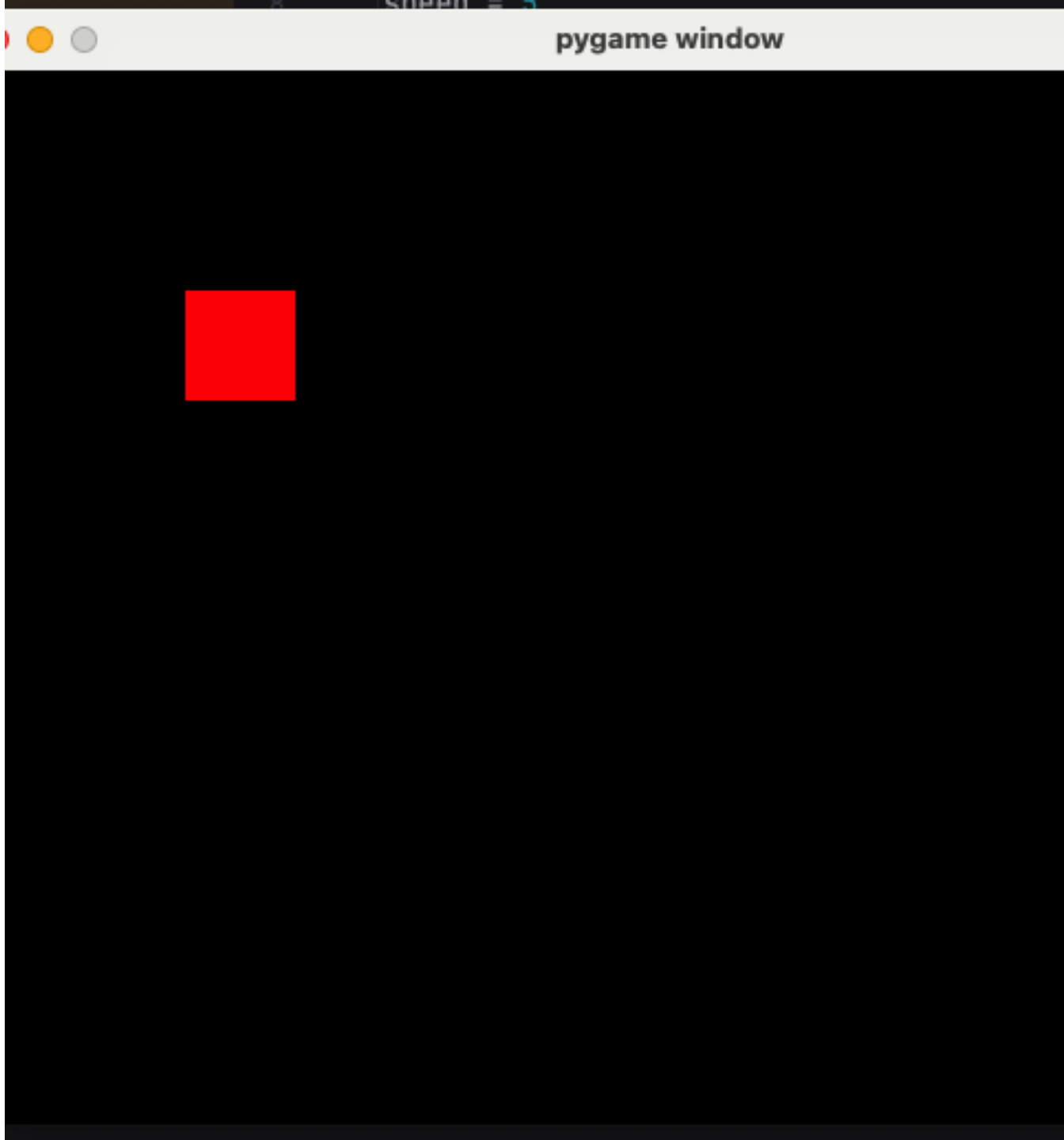
```
x = 100  # początkowa pozycja X
y = 100  # początkowa pozycja Y
speed = 5  # prędkość ruchu
```

```
keys = pygame.key.get_pressed()
if keys[pygame.K_LEFT]:
    x -= speed
if keys[pygame.K_RIGHT]:
    x += speed
if keys[pygame.K_UP]:
    y -= speed
if keys[pygame.K_DOWN]:
    y += speed
```

```
import pygame
pygame.init()

screen = pygame.display.set_mode((640, 480))
clock = pygame.time.Clock()

x, y = 100, 100
speed = 5
running = True

while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    keys = pygame.key.get_pressed()
    if keys[pygame.K_LEFT]:
        x -= speed
    if keys[pygame.K_RIGHT]:
        x += speed
    if keys[pygame.K_UP]:
        y -= speed
    if keys[pygame.K_DOWN]:
        y += speed

    screen.fill((0, 0, 0))  # czarne tło
    pygame.draw.rect(screen, (255, 0, 0), (x, y, 50, 50))  # czerwony
kwadrat jako postać

    pygame.display.flip()
    clock.tick(60)  # 60 FPS

pygame.quit()
```

Ruch postaci w PyGame

Efekt? Za pomocą klawiszy przesuwamy czerwony kwadrat. Ciekawostką jest ilość klatek na sekundę oraz zastosowanie Canvas do wyrysowania postaci. Zamiast niej może być element określany CharCode lub wyrysowany ze zdjęcia, któremu nadamy możliwość animacji.



Gry w okienkowym Pythonie

Gra „Snake”

```
import tkinter as tk
import random

WIDTH, HEIGHT = 400, 400
SEG_SIZE = 20

class SnakeGame:
  def __init__(self, root):
    self.root = root
    self.direction = 'Right'
    self.score = 0

    self.canvas = tk.Canvas(root, width=WIDTH, height=HEIGHT, bg='black')
    self.canvas.pack()

    self.segments = [(SEG_SIZE*5, SEG_SIZE*5)]
    self.food = None
    self.create_food()

    # Sterowanie klawiszami ADWS
    root.bind('a', self.go_left)
    root.bind('d', self.go_right)
    root.bind('w', self.go_up)
    root.bind('s', self.go_down)

    self.move_snake()

  def create_food(self):
    while True:
      x = random.randint(0, (WIDTH//SEG_SIZE)-1)*SEG_SIZE
      y = random.randint(0, (HEIGHT//SEG_SIZE)-1)*SEG_SIZE
      if (x, y) not in self.segments:
        self.food = (x, y)
        break

  def move_snake(self):
    head_x, head_y = self.segments[-1]

    if self.direction == 'Left':
      head_x -= SEG_SIZE
    elif self.direction == 'Right':
      head_x += SEG_SIZE
    elif self.direction == 'Up':
      head_y -= SEG_SIZE
    elif self.direction == 'Down':
      head_y += SEG_SIZE

    new_head = (head_x, head_y)

    # Sprawdzenie kolizji
    if (head_x < 0 or head_x >= WIDTH or head_y < 0 or head_y >= HEIGHT or
      new_head in self.segments):
      self.game_over()
      return

    self.segments.append(new_head)

    if new_head == self.food:
      self.score += 1
      self.create_food()
    else:
      self.segments.pop(0)

    self.draw()
    self.root.after(400, self.move_snake)  # Ruch co 400 ms

  def draw(self):
    self.canvas.delete('all')
    for x, y in self.segments:
      self.canvas.create_rectangle(x, y, x+SEG_SIZE, y+SEG_SIZE, fill='green')
    fx, fy = self.food
    self.canvas.create_oval(fx, fy, fx+SEG_SIZE, fy+SEG_SIZE, fill='red')
    self.canvas.create_text(50, 10, text=f"Score: {self.score}", fill="white",
anchor='nw')

  def game_over(self):
    self.canvas.delete('all')
    self.canvas.create_text(WIDTH//2, HEIGHT//2, text=f"Koniec gry! Wynik:
{self.score}",
                fill='red', font=('Arial', 24))

  def go_left(self, event):
    if self.direction != 'Right':
      self.direction = 'Left'
  def go_right(self, event):
    if self.direction != 'Left':
      self.direction = 'Right'
  def go_up(self, event):
    if self.direction != 'Down':
      self.direction = 'Up'
  def go_down(self, event):
    if self.direction != 'Up':
      self.direction = 'Down'

root = tk.Tk()
root.title("Prosty Snake - Sterowanie ADWS")
game = SnakeGame(root)
root.mainloop()
```

Gry w obiektowym Pythonie

bind

Metoda bind w Tkinter służy do **powiązania zdarzeń z funkcjami (handlerami)**, które mają zostać wywołane, gdy wystąpi dane zdarzenie w oknie lub widżecie.

W kontekście gry Snake:

root.bind('a', self.go_left)

- oznacza, że **gdy użytkownik naciśnie klawisz „a”**, zostanie wywołana metoda go_left, która zmienia kierunek ruchu węża.
- inne przykłady zdarzeń to kliknięcia myszą, naciśnięcia innych klawiszy, ruchy kursora itd.

Gry w obiektowym Pythonie

SEG_SIZE

SEG_SIZE to stała określająca **rozmiar pojedynczego segmentu węża i jednocześnie jednostkę ruchu**.

W grze Snake:

- Każdy segment węża ma rozmiar kwadratu o boku SEG_SIZE (np. 20 pikseli).
- Wąż przesuwa się o dokładnie tę odległość przy każdym ruchu, dzięki czemu jego segmenty idealnie „stawiają kroki” na siatce o wymiarach SEG_SIZE x SEG_SIZE.
- Ułatwia to detekcję kolizji i rysowanie segmentów w stałych, równych odstępach.

Gry w obiektowym Pythonie

Wiele poziomów w grze?

Możesz przygotować osobne konfiguracje lub funkcje dla każdego poziomu, np.:

różne szybkości węża (mniejszy after — szybszy ruch),

inne rozmiary planszy,

więcej przeszkód na planszy,

inne cele lub liczba punktów do zdobycia.

Gry w obiektowym Pythonie

Potrzebna jest zmienna przechowująca każdy z poziomów:

```
self.level = 1
```

Wywołanie każdego kolejnego poziomu:

```
self.level += 1
self.load_level(self.level)
```

Metoda wywołania każdego kolejnego poziomu:

```
def load_level(self, level):
    if level == 1:
        self.speed = 400
        # inne ustawienia poziomu 1
    elif level == 2:
        self.speed = 300
        # inne ustawienia poziomu 2, np. przeszkody
    elif level == 3:
        self.speed = 200
        # itd.
```

W move_snake zamiast stałej wartości:

```
self.root.after(self.speed, self.move_snake)
```

Gry w obiektowym Pythonie

Warunek przejścia do kolejnego poziomu:

```
if self.score == 5:
    self.level += 1
    self.load_level(self.level)
```

Możesz również zrobić zmianę planszy:

- Dodaj na planszy **przeszkody** (prostokąty, których wąż nie może dotknąć),
- zmień rozmiar planszy,
- zmień wygląd lub zachowanie węża.

Gry w obiektowym Pythonie

Schemat dodania poziomu:

```
class SnakeGame:
    def __init__(self, root):
        self.level = 1
        self.speed = 400
        # reszta inicjalizacji...

    def load_level(self, level):
        if level == 1:
            self.speed = 400
            # np. brak przeszkód
        elif level == 2:
            self.speed = 300
            self.obstacles = [(100, 100), (120, 100)] # przykładowe przeszkody
        # itp.

    def move_snake(self):
        # ruch węża...
        if self.score == 5 and self.level == 1:
            self.level = 2
            self.load_level(2)
        # ruch po przeszkodach, sprawdzanie kolizji itd.

        self.root.after(self.speed, self.move_snake)
```

Gry w obiektowym Pythonie

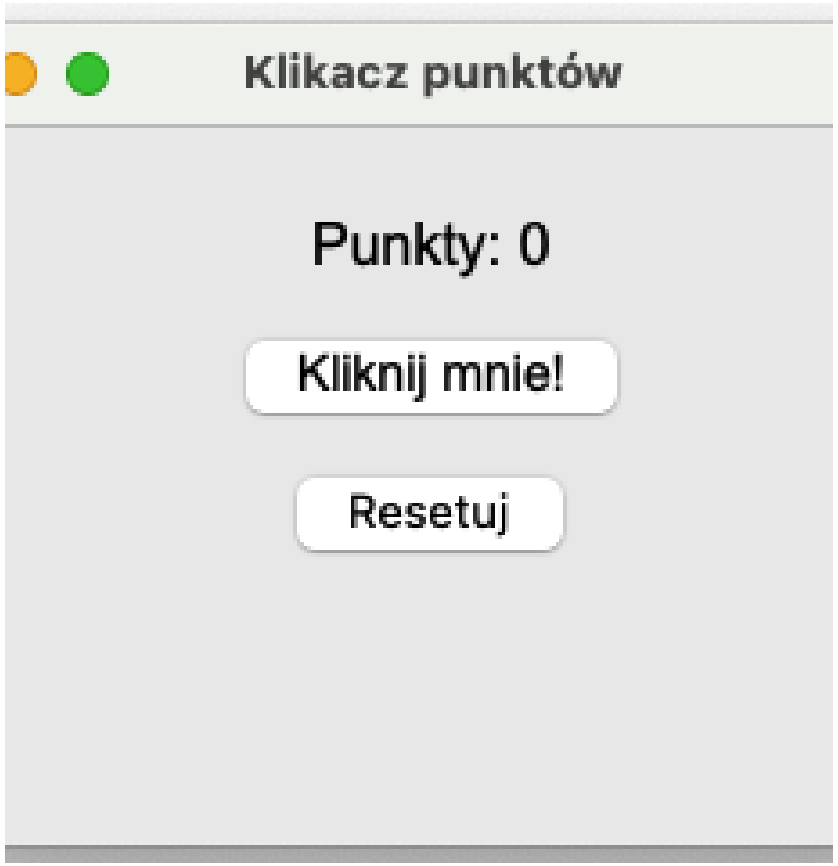
Gra rozpisana na kilka plików zwykle zawiera:

- uruchamianie gry w pliku main.py
- logikę gry w kolejnych plikach
- połączenie plików odbywa się poprzez import klas.

```
# main.py

import tkinter as tk
from game import Game

def main():
    root = tk.Tk()
    root.title("Klikacz punktów")
    root.geometry("300x200")

    gra = Game(root)

    root.mainloop()

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

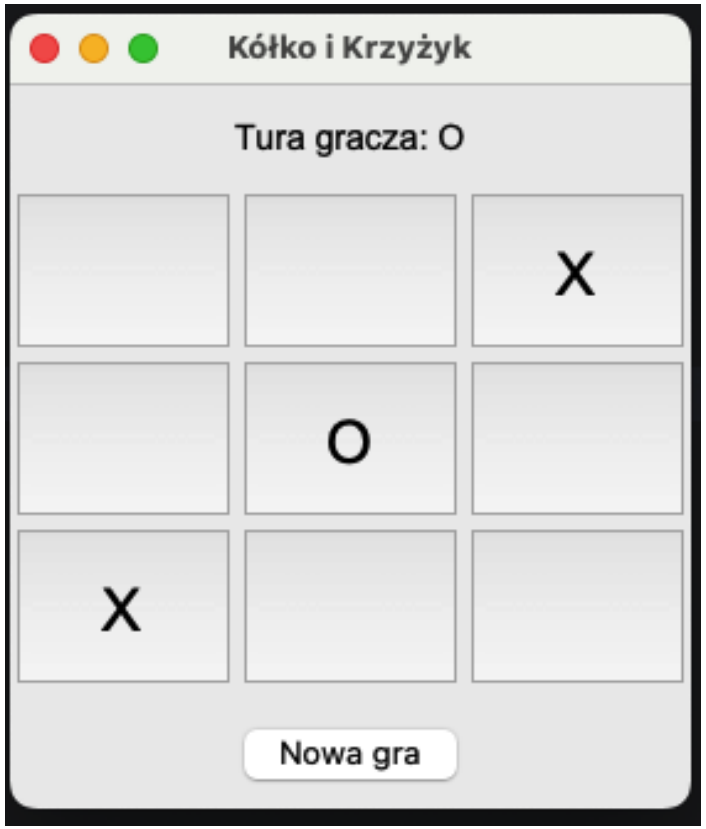
Gry w obiektowym Pythonie

```
import tkinter as tk

class Game:
    def __init__(self, master):
        self.master = master
        self.points = 0

        self.frame = tk.Frame(master)
        self.frame.pack(padx=20, pady=20)

        self.label = tk.Label(self.frame, text=f"Punkty: {self.points}", font=("Arial", 16))
        self.label.pack()

        self.button = tk.Button(self.frame, text="Kliknij mnie!", font=("Arial", 14), command=self.increase_points)
        self.button.pack(pady=10)

        self.reset_button = tk.Button(self.frame, text="Resetuj", command=self.reset_points)
        self.reset_button.pack()

    def increase_points(self):
        self.points += 1
        self.label.config(text=f"Punkty: {self.points}")

    def reset_points(self):
        self.points = 0
        self.label.config(text=f"Punkty: {self.points}")
```



Gry w obiektowym Pythonie

Gra „Kółko i krzyżyk”



Struktura gry jest rozpisana w trzech plikach:

- main.py uruchamia grę
- game.py zawiera logikę gry
- gui.py zawiera wygląd gry.

Gry w obiektowym Pythonie

```
# game.py

class TicTacToeGame:
    def __init__(self):
        self.reset_game()

    def reset_game(self):
        self.board = [["" for _ in range(3)] for _ in range(3)]
        self.current_player = "X"
        self.game_over = False

    def make_move(self, row, col):
        if self.board[row][col] == "" and not self.game_over:
            self.board[row][col] = self.current_player
            if self.check_winner(row, col):
                self.game_over = True
                return f"{self.current_player} wygrał!"
            elif self.is_draw():
                self.game_over = True
                return "Remis!"
            else:
                self.current_player = "O" if self.current_player == "X" else "X"
                return None
        return None

    def check_winner(self, row, col):
        b = self.board
        p = self.current_player

        # Wiersz, kolumna, przekątne
        return (
            all(b[row][c] == p for c in range(3)) or
            all(b[r][col] == p for r in range(3)) or
            all(b[i][i] == p for i in range(3)) or
            all(b[i][2 - i] == p for i in range(3))
        )

    def is_draw(self):
        return all(self.board[r][c] != "" for r in range(3) for c in range(3))
```

Zawiera funkcje do resetu gry oraz ruchu po siatce, w której umieszczono znaki.

Sprawdzenie wyniku odbywa się poprzez operacje na kilku zmiennych i pętle.

Gry w obiektowym Pythonie

```
# gui.py

import tkinter as tk
from game import TicTacToeGame


class TicTacToeGUI:
    def __init__(self, root):
        self.root = root
        self.root.title("Kółko i Krzyżyk")
        self.game = TicTacToeGame()

        self.buttons = [[None for _ in range(3)] for _ in range(3)]
        self.status_label = tk.Label(root, text="Gracz X zaczyna", font=("Arial", 14))
        self.status_label.pack(pady=10)

        self.board_frame = tk.Frame(root)
        self.board_frame.pack()

        for r in range(3):
            for c in range(3):
                btn = tk.Button(self.board_frame, text="", font=("Arial", 24), width=4, height=2,
                                command=lambda r=r, c=c: self.on_click(r, c))
                btn.grid(row=r, column=c)
                self.buttons[r][c] = btn

        self.reset_button = tk.Button(root, text="Nowa gra", command=self.reset_game)
        self.reset_button.pack(pady=10)


    def on_click(self, row, col):
        result = self.game.make_move(row, col)
        self.update_ui()

        if result:
            self.status_label.config(text=result)


    def update_ui(self):
        for r in range(3):
            for c in range(3):
                self.buttons[r][c].config(text=self.game.board[r][c])

        if not self.game.game_over:
            self.status_label.config(text=f"Tura gracza: {self.game.current_player}")


    def reset_game(self):
        self.game.reset_game()
        self.update_ui()
        self.status_label.config(text="Gracz X zaczyna")
```

Zawiera okno oraz formatowanie elementów.

Uruchomienie programu.

```
# main.py

import tkinter as tk
from gui import TicTacToeGUI

def main():
    root = tk.Tk()
    app = TicTacToeGUI(root)
    root.mainloop()

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

Multimedia
w oknie Tkinter

Do ich obsługi potrzebujemy kontrolek tych samych co w grach (Pygame).

Dla potrzeb estetycznych warto zastosować dodatkowe biblioteki do formatowania.

Z prawej strony przykład wykorzystania. Przed Wami odtwarzacz MP3 pobierający muzykę z folderu i wyświetlający listę plików.

Odtwarzacz MP3 – przykład zastosowania

```
import os
import tkinter as tk
from tkinter import filedialog
from ttkbootstrap import Style
from ttkbootstrap.constants import *
from tkinter import ttk
import pygame


class MP3Player:
    def __init__(self, root):
        self.root = root
        self.root.title("Odtwarzacz MP3")
        self.style = Style("superhero")  #inne motywy: "cosmo", "minty", "darkly", "journal", "cyborg", "flatly", itd.

        # Inicjalizacja pygame mixer
        pygame.mixer.init()

        self.playlist = []
        self.current_index = -1
        self.paused = False

        # Interfejs
        self.label = ttk.Label(root, text="Brak wybranego folderu", font=("Arial", 12))
        self.label.pack(pady=10)

        self.listbox = tk.Listbox(root, width=50)
        self.listbox.pack(pady=10)
        self.listbox.bind("<<ListboxSelect>>", self.select_song)

        btn_frame = ttk.Frame(root)
        btn_frame.pack(pady=10)

        ttk.Button(btn_frame, text="🎶 Wybierz folder", command=self.load_folder, bootstyle=PRIMARY).grid(row=0, column=0, padx=5)

        ttk.Button(btn_frame, text="▶️ Odtwórz", command=self.play_song, bootstyle=SUCCESS).grid(row=0, column=1, padx=5)

        ttk.Button(btn_frame, text="⏸️ Pauza", command=self.pause_song, bootstyle=WARNING).grid(row=0, column=2, padx=5)

        ttk.Button(btn_frame, text="⏹️ Stop", command=self.stop_song, bootstyle=DANGER).grid(row=0, column=3, padx=5)

        self.now_playing = ttk.Label(root, text="Aktualnie: brak", font=("Arial", 10, "italic"))
        self.now_playing.pack(pady=5)

    def load_folder(self):
        folder = filedialog.askdirectory()
        if folder:
            self.playlist = [f for f in os.listdir(folder) if f.endswith(".mp3")]
            self.folder_path = folder
            self.listbox.delete(0, tk.END)
            for song in self.playlist:
                self.listbox.insert(tk.END, song)
            self.label.config(text=f"Wybrany folder: {os.path.basename(folder)}")
            self.current_index = -1
            self.now_playing.config(text="Aktualnie: brak")

    def select_song(self, event):
        selected = self.listbox.curselection()
        if selected:
            self.current_index = selected[0]
            self.play_song()

    def play_song(self):
        if self.current_index == -1 and self.playlist:
            self.current_index = 0
        if self.current_index >= 0:
            file_path = os.path.join(self.folder_path, self.playlist[self.current_index])
            pygame.mixer.music.load(file_path)
            pygame.mixer.music.play()
            self.now_playing.config(text=f"Aktualnie: {self.playlist[self.current_index]}")
            self.paused = False

    def pause_song(self):
        if pygame.mixer.music.get_busy():
            if self.paused:
                pygame.mixer.music.unpause()
                self.paused = False
            else:
                pygame.mixer.music.pause()
                self.paused = True

    def stop_song(self):
        pygame.mixer.music.stop()
        self.now_playing.config(text="Aktualnie: brak")
        self.paused = False


# Uruchomienie aplikacji
if __name__ == "__main__":
    root = tk.Tk()
    app = MP3Player(root)
    root.mainloop()
```

Odtwarzacz video w Pythonie

Dwa sposoby:

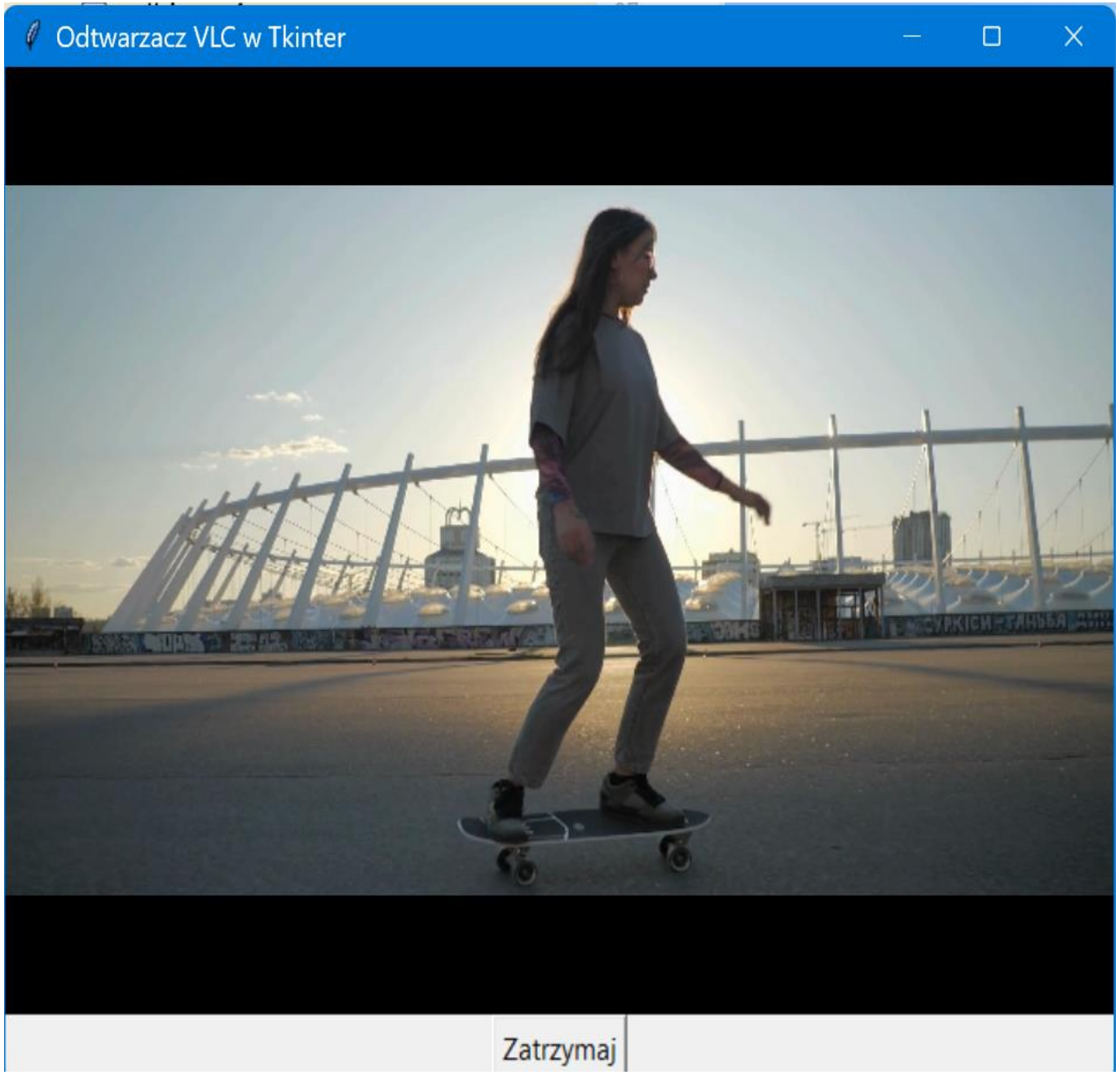
- Tkintervideo
- VLC

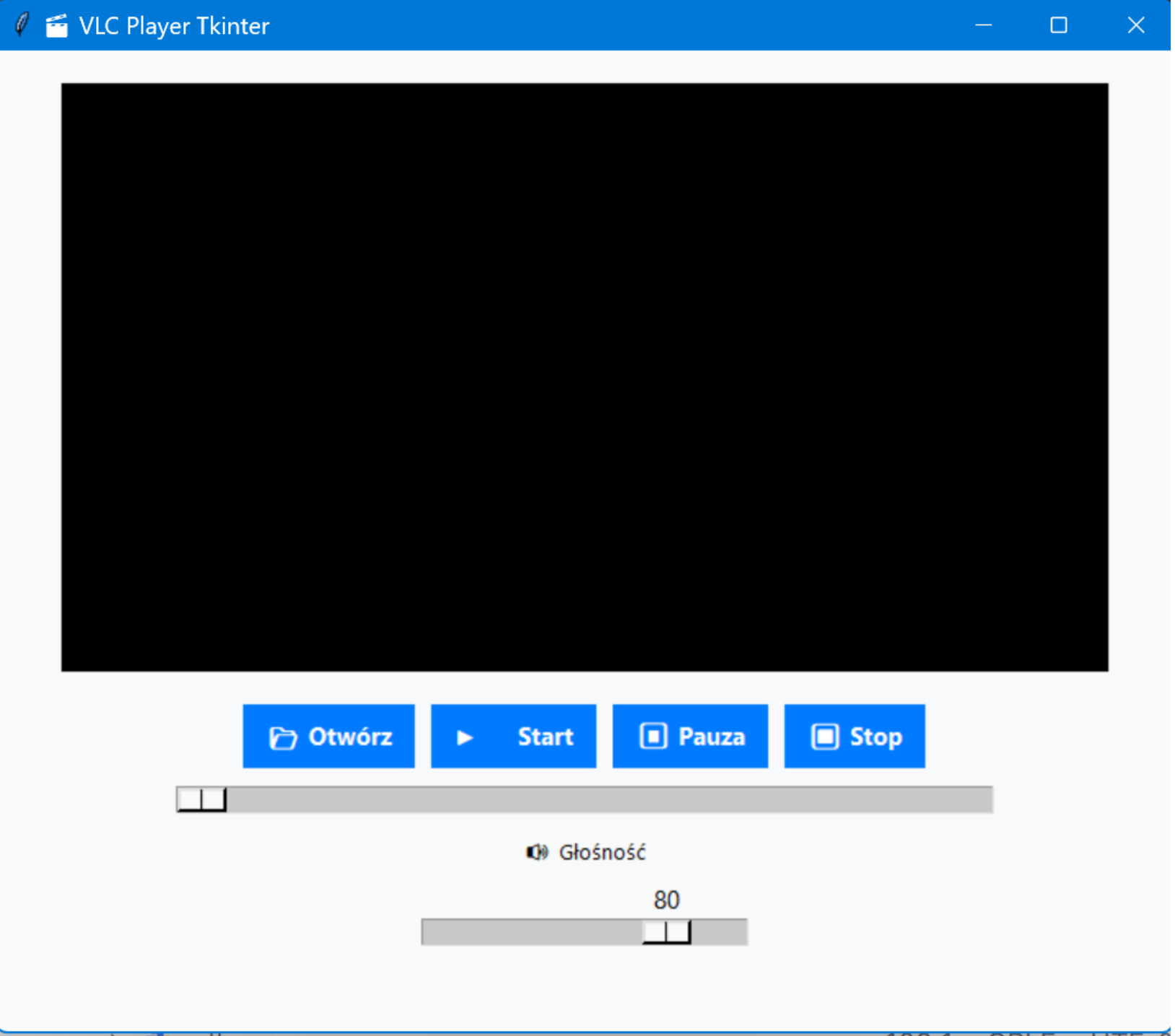
Wykorzystanie kodeków z odtwarzacza VLC to prostszy sposób. Wystarczy zainstalować program do odtwarzania oraz dołączyć odpowiednią instancję do projektu:

```
instance = vlc.Instance()
```

Przykład zastosowania

```
import tkinter as tk
import vlc

# Stwórz okno Tkinter
root = tk.Tk()
root.title("Odtwarzacz VLC w Tkinter")

# Stwórz instancję VLC
instance = vlc.Instance()

# Stwórz kontroler mediów
player = instance.media_player_new()

# Dodaj Frame do okna
my_frame = tk.Frame(root, width=640,
height=480)
my_frame.pack()

# Utwórz identyfikator okna
my_window_id = my_frame.winfo_id()

# Ustaw preferowany interfejs
player.set_hwnd(my_window_id)

# Załaduj plik multimedialny
media =
instance.media_new("rolki.mp4")
player.set_media(media)

# Odtwarzaj
player.play()

# Przycisk do zatrzymania
button_stop = tk.Button(root,
text="Zatrzymaj", command=player.stop)
button_stop.pack()

# Uruchom pętlę zdarzeń Tkinter
root.mainloop()
```



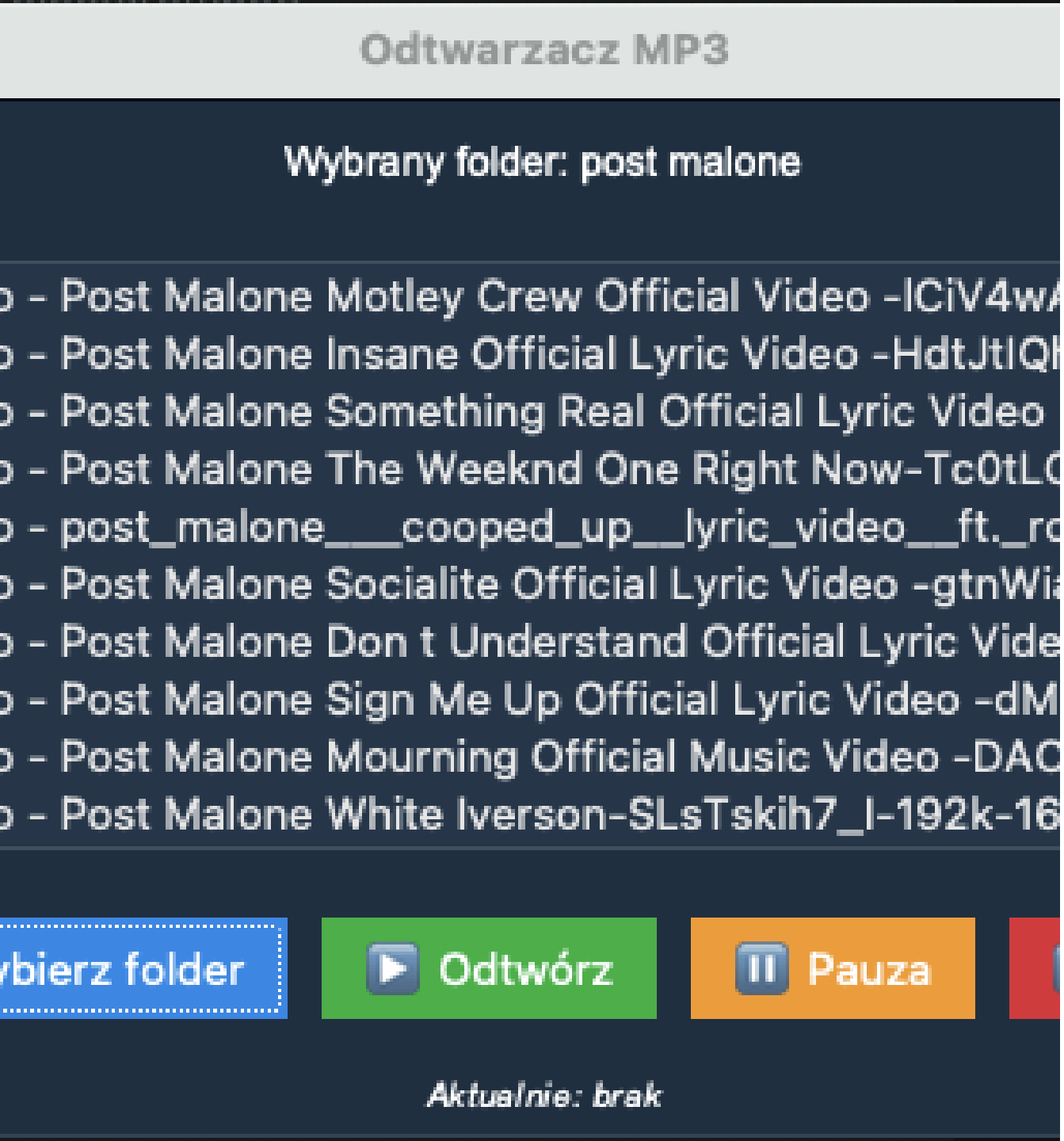


Możliwości rozszerzenia:

- Dodatkowe kontrolki
- Menu z górnej części okna
- Wybór filmów z listy

Przykładowy kod rozszerzenia odtwarzacza

```
import tkinter as tk
import vlc
import platform

# --- Stylizacja Bootstrap-like ---
BUTTON_FONT = ("Segoe UI", 12, "bold")
BUTTON_STYLE = {
    "padx": 10,
    "pady": 5,
    "bd": 0,
    "fg": "white",
    "bg": "#007BFF",     # Niebieski jak w Bootstrap
    "activebackground": "#0056b3",
    "font": BUTTON_FONT,
    "cursor": "hand2"
}

# --- Tworzenie GUI ---
root = tk.Tk()
root.title("🎬 Odtwarzacz VLC w Tkinter")
root.geometry("700x550")
root.configure(bg="#f8f9fa")  # Jasne tło jak Bootstrap

# VLC
instance = vlc.Instance()
player = instance.media_player_new()

# Frame na video
video_frame = tk.Frame(root, width=640, height=360,
bg="black")
video_frame.pack(pady=20)

root.update()  # Potrzebne, by uzyskać poprawny ID okna
window_id = video_frame.winfo_id()

# Dopasowanie do systemu
system = platform.system()
if system == 'Windows':
    player.set_hwnd(window_id)
elif system == 'Linux':
    player.set_xwindow(window_id)
elif system == 'Darwin':  # macOS
    player.set_nsobject(window_id)

# Media
media = instance.media_new("rolki.mp4")
player.set_media(media)

# --- Funkcje ---
def play():
    player.play()

def pause():
    player.pause()

def stop():
    player.stop()

def louder():
    current_volume = player.audio_get_volume()
    player.audio_set_volume(min(current_volume + 10, 100))

def quieter():
    current_volume = player.audio_get_volume()
    player.audio_set_volume(max(current_volume - 10, 0))

# --- Przyciski ---
controls = tk.Frame(root, bg="#f8f9fa")
controls.pack()

btn_play = tk.Button(controls, text="▶️ Start",
command=play, **BUTTON_STYLE)
btn_pause = tk.Button(controls, text="⏸️ Pauza",
command=pause, **BUTTON_STYLE)
btn_stop = tk.Button(controls, text="⏹️ Stop",
command=stop, **BUTTON_STYLE)
btn_vol_up = tk.Button(controls, text="🔊 +",
command=louder, **BUTTON_STYLE)
btn_vol_down = tk.Button(controls, text="🔉 -",
command=quieter, **BUTTON_STYLE)

# Ustawienie w siatce
btn_play.grid(row=0, column=0, padx=5, pady=5)
btn_pause.grid(row=0, column=1, padx=5, pady=5)
btn_stop.grid(row=0, column=2, padx=5, pady=5)
btn_vol_down.grid(row=0, column=3, padx=5, pady=5)
btn_vol_up.grid(row=0, column=4, padx=5, pady=5)

# Uruchomienie GUI
root.mainloop()
```



Winamp w Tkinter?

Czego potrzebujesz do takiego projektu?

Bibliotek matematycznych – numpy, matplotlib

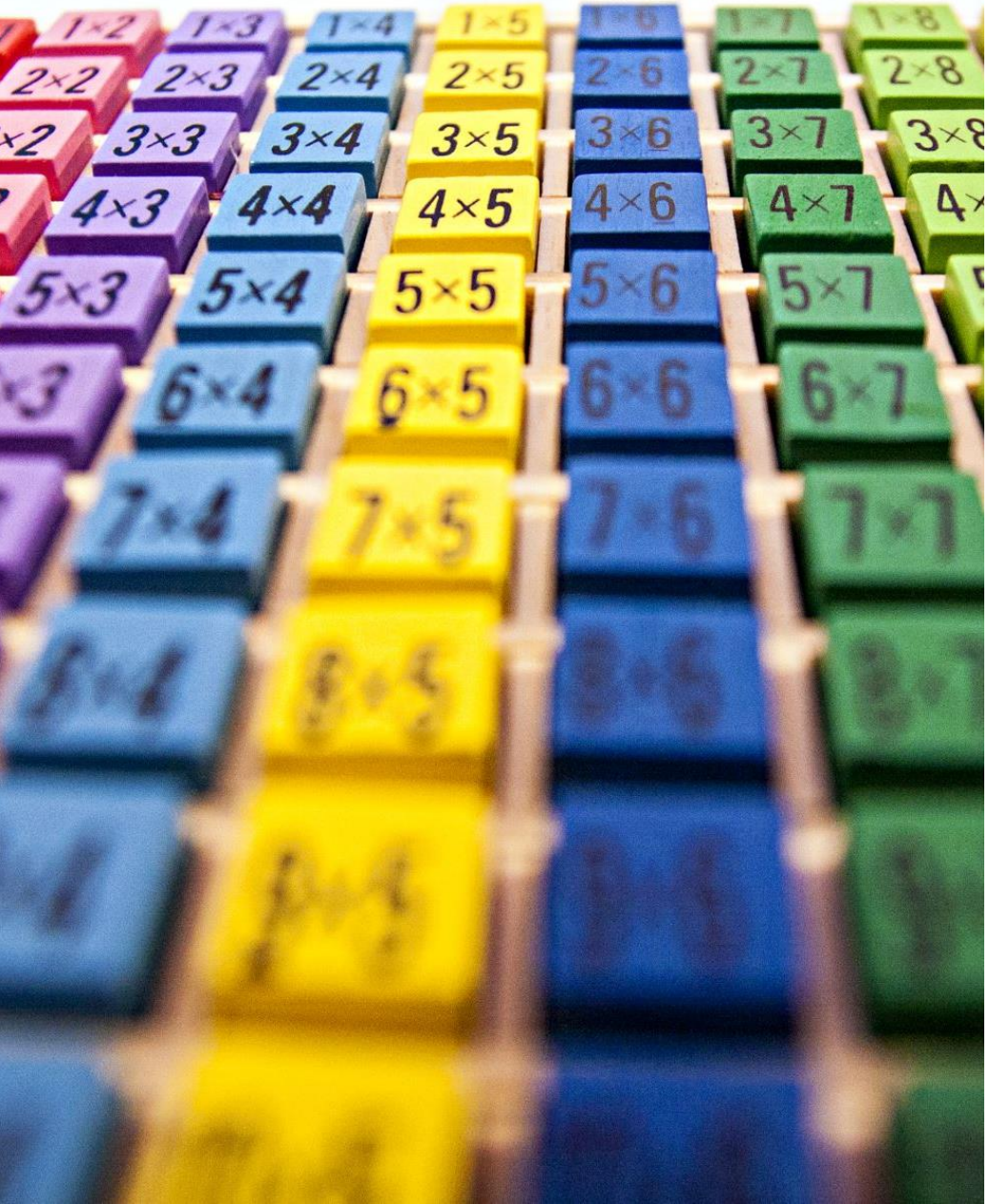
Bilibotek muzycznych – pydub

Bibliotek do zarządzania czasem – threadin i time

Bibliotek audiowizualnych – vlc.

To absolutne minimum.

Do tego będą potrzebne ikony, listy, Bootstap oraz inne według uznania. Uwaga na kodeki audio I video, które często powodują problem podczas otwarzania.



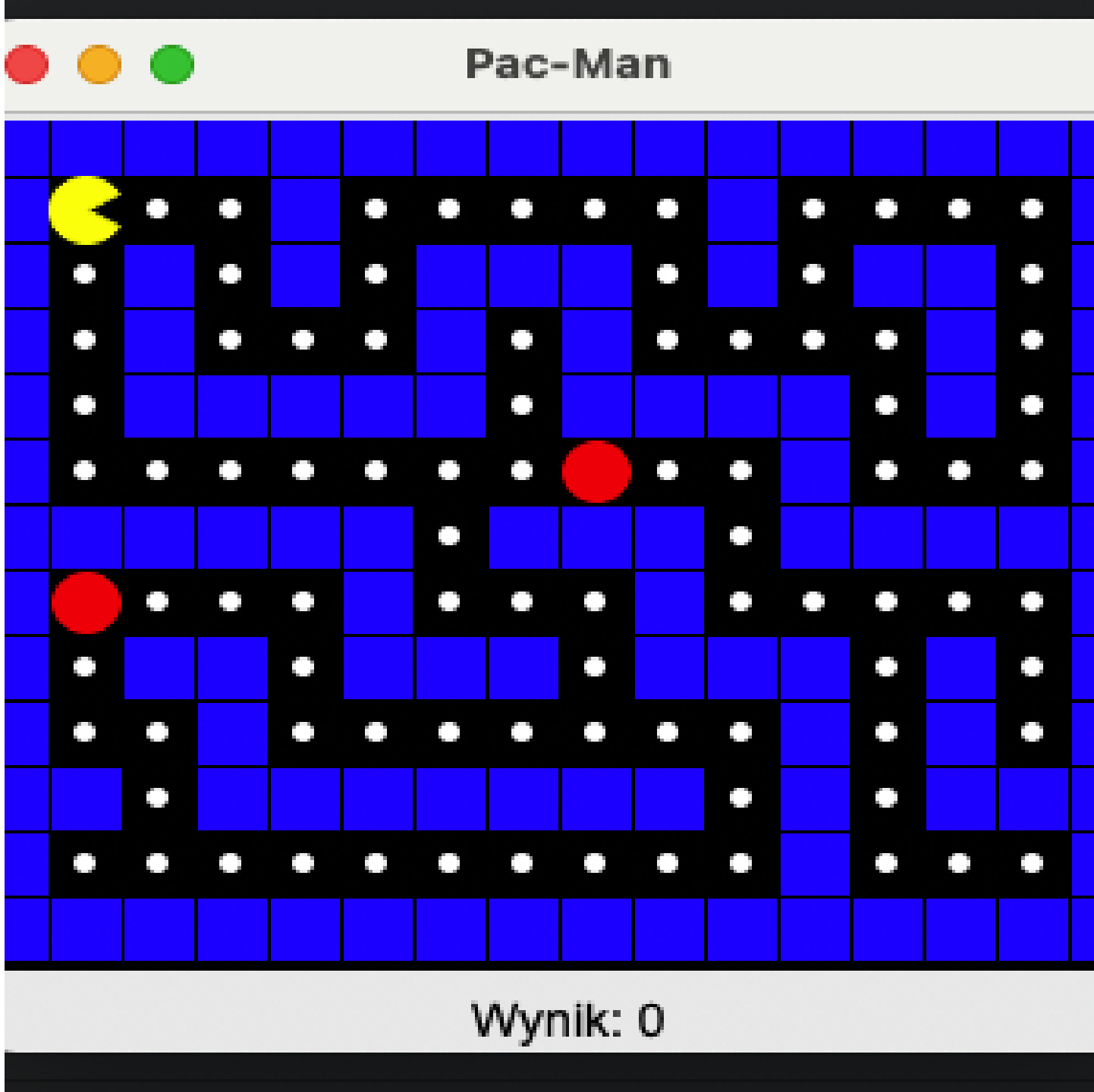
Podsumowanie - pytania

1. Czy biblioteki matematyczne są potrzebne do tworzenia gier?
2. W jaki sposób tworzy się wiele planszy jednej gry?
3. Co można zrobić za pomocą Bootstrap w języhku Python?
4. Jakiego rodzajów kodeków należy użyć w odtwarzaczu multimedialnym?
5. Wymień znane Tobie biblioteki muzyczne?
6. Co można zrobić za pomocą bibliotek matematycznych w projekcie odtwarzacza?

Gry do samodzielnego wykonania

Zadanie 1

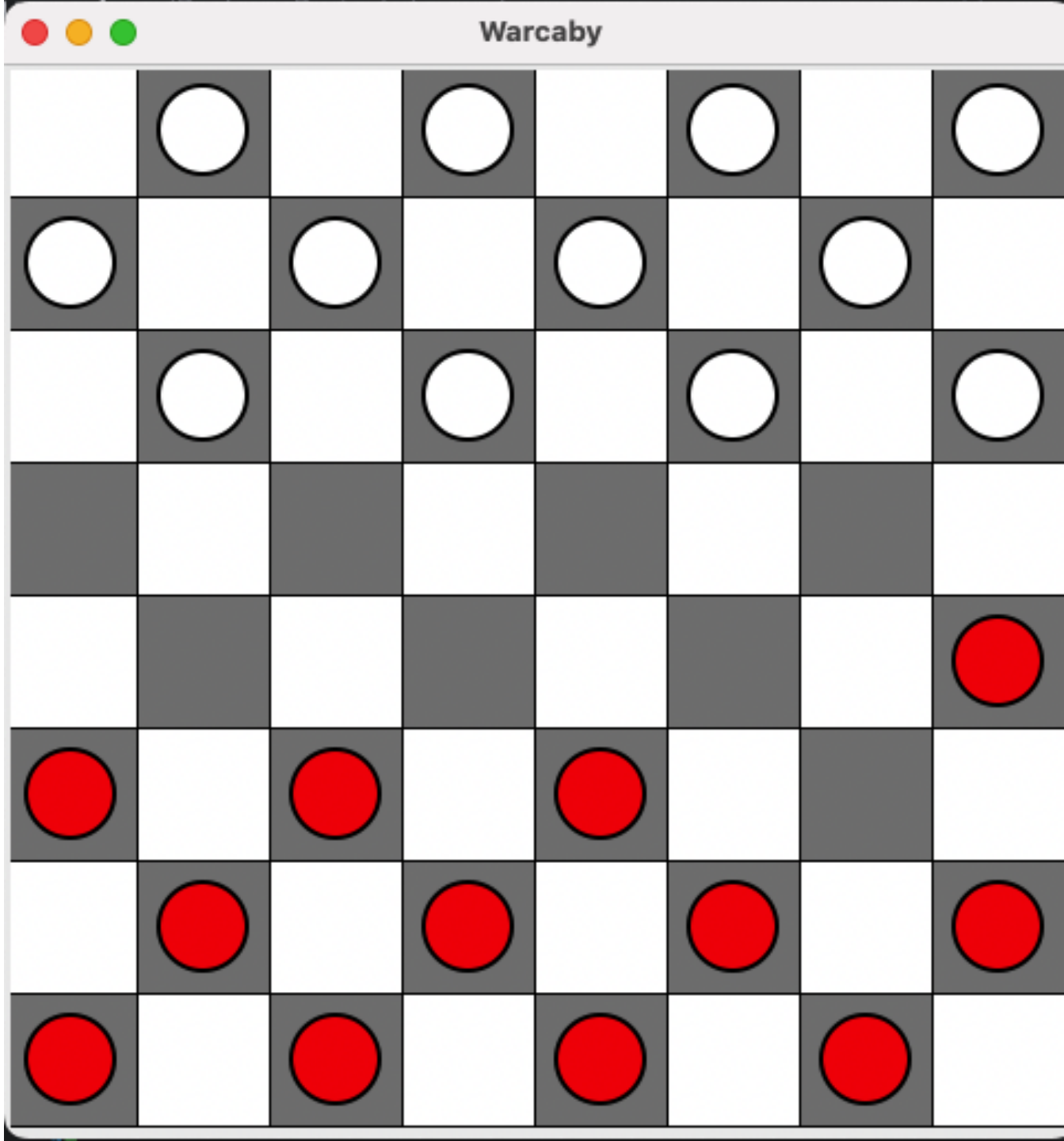
Gra „Pac-Man”



Gry do samodzielnego wykonania

Zadanie 2

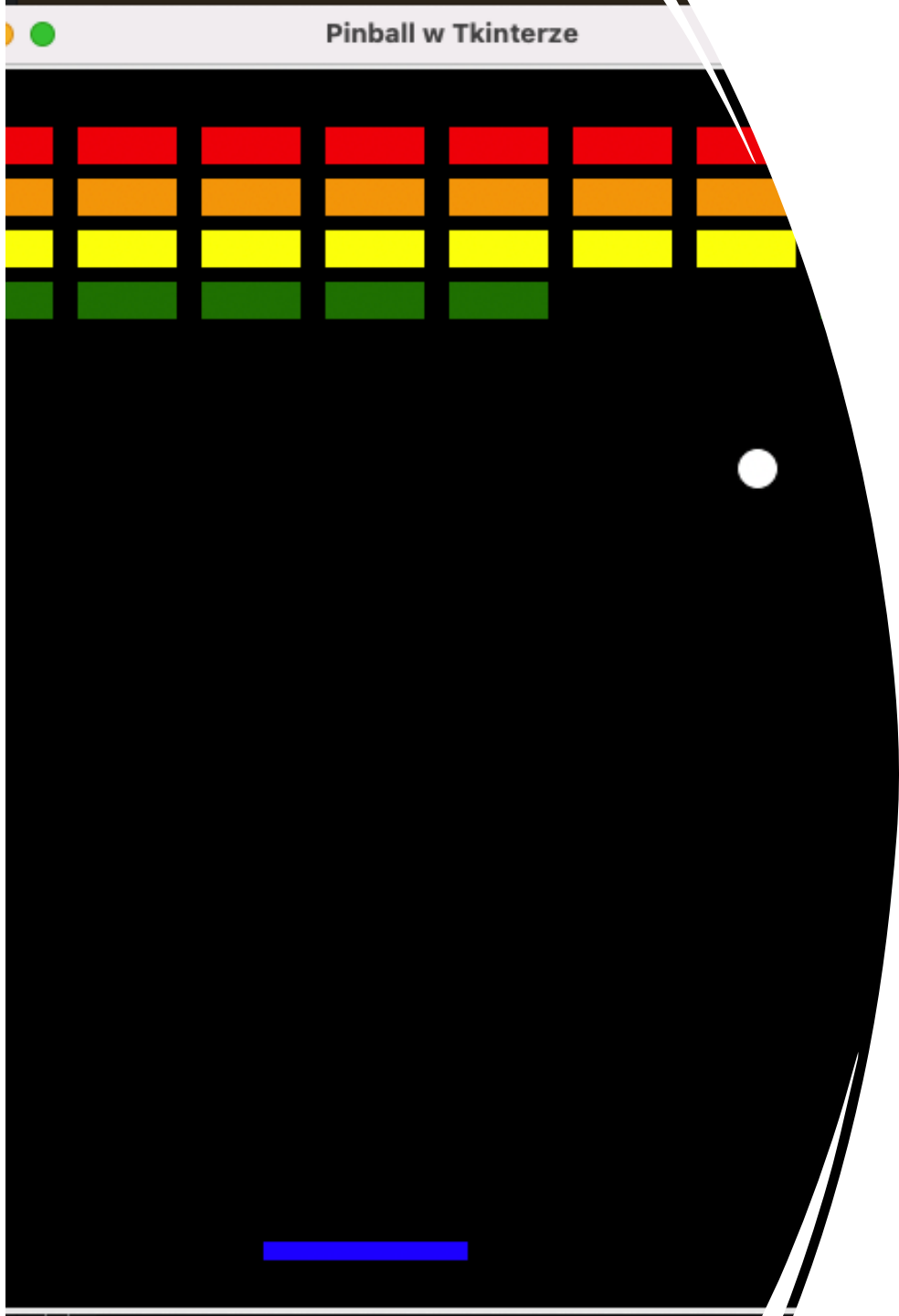
Warcaby

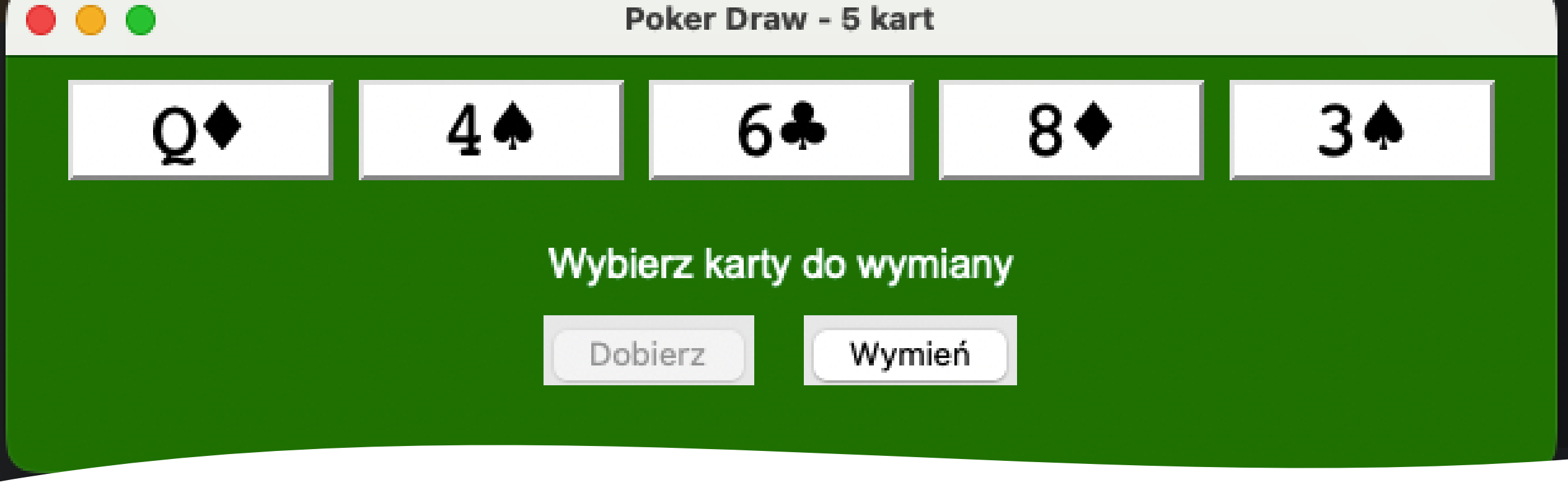


Gry do samodzielnego wykonania

Zadanie 3

Gra „Pinball”





Gry do samodzielnego wykonania

Zadanie 4

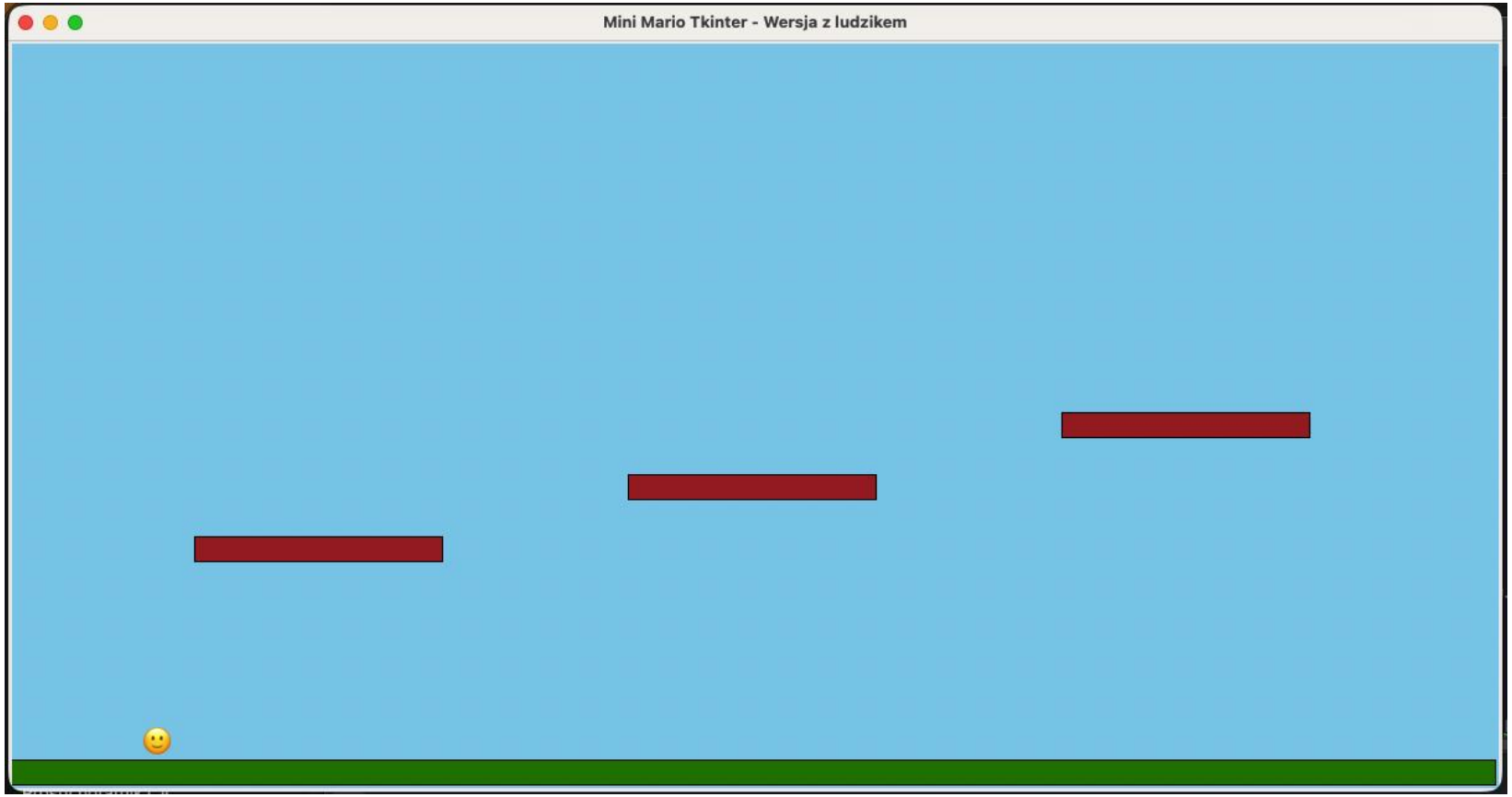
Gra „Poker” na 5 kart

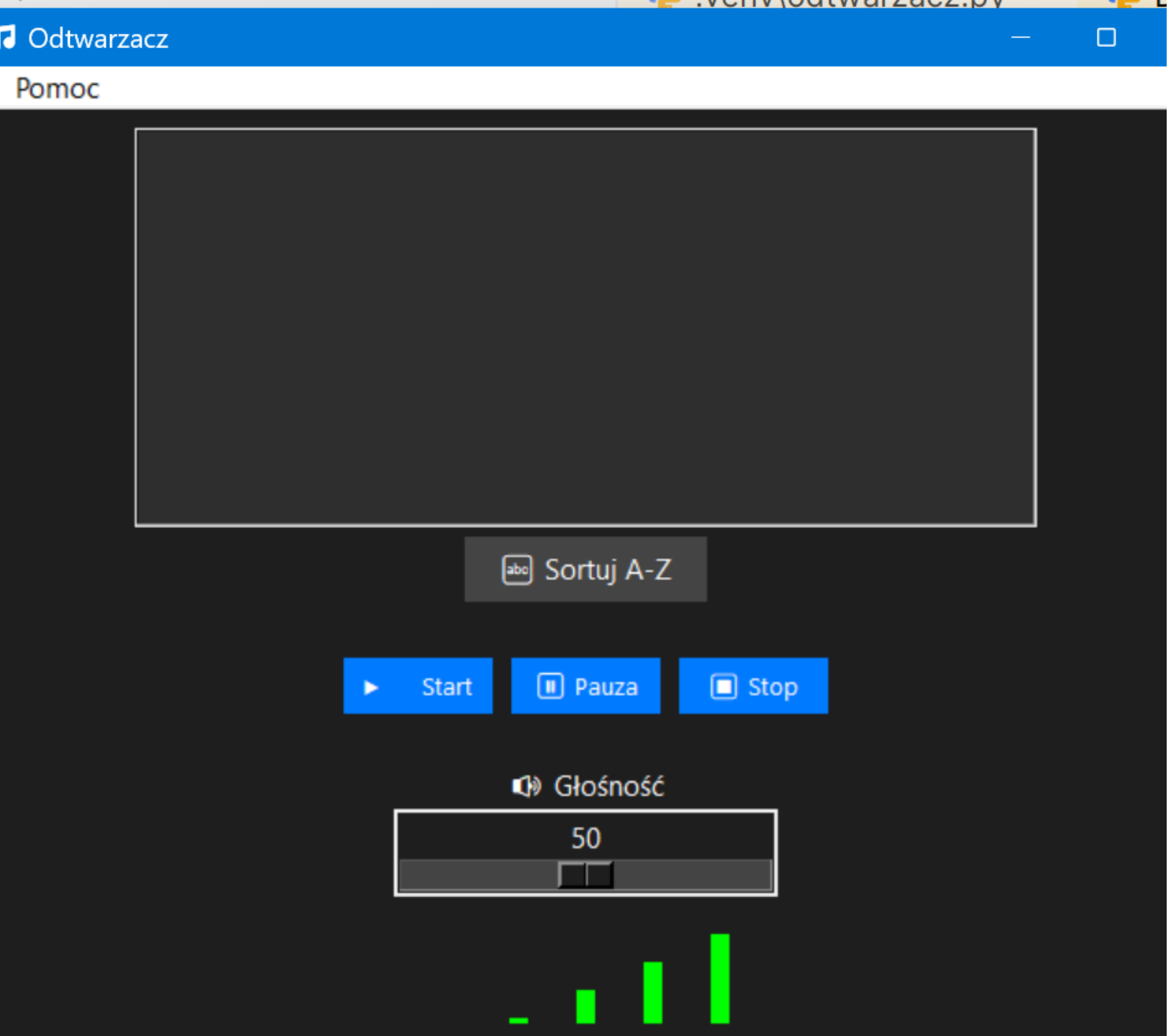
Gry do samodzielnego wykonania

Zadanie 5

Gra „Skacząca buźka”?

Tak, to dopiero początek gry Mario. Dodaj do niej kilka plansz
i przeciwników.





Odtwarzacz do samodzielnego wykonania

- Celem zadania jest projekt i wykonanie odtwarzacza MP3, który pobiera tytuły z katalogu, wyświetla je w liście i pozwala je posortować alfabetycznie.

- Do obsługi służą trzy przyciski. Głośność regulowana jest za pomocą suwaka w zakresie 0-100. Pod suwakiem znajduje się prosty equalizer w postaci pasków.

Wykorzystanie Pythona w AI

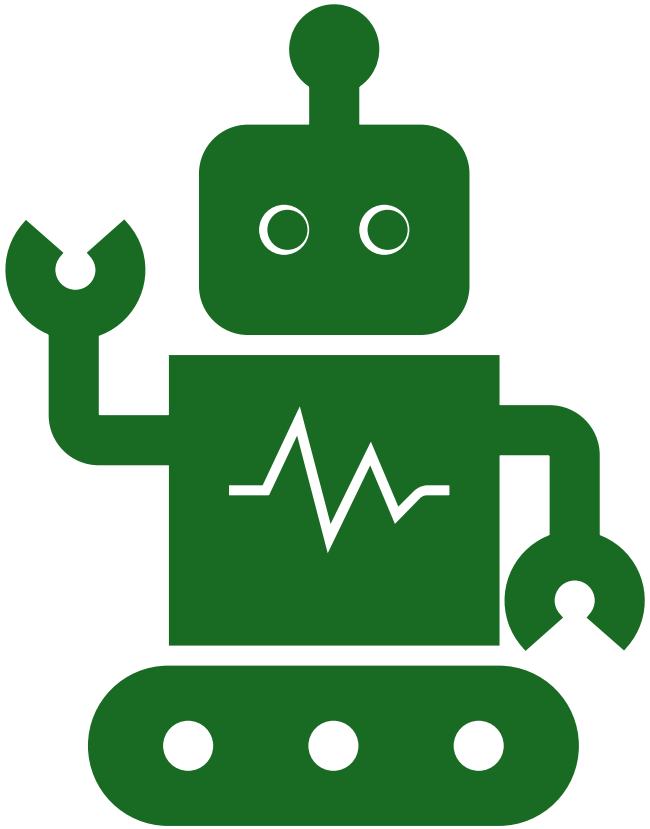
- Asystenci głosowi (Siri, Alexa)
- Filtry spamu i antywirusy
- Tłumacze językowe (np. Google Translate)
- Personalizowane reklamy i oferty
- Medycyna – analiza obrazów RTG, diagnozy

Wykorzystanie Pythona w AI

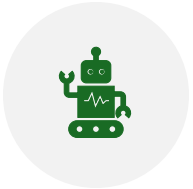
Cecha	Uczenie maszynowe (ML)	Głębokie uczenie (DL)
Dane	Mniej danych	Dużo danych
Czas treningu	Krótszy	Dłuższy
Model	np. drzewa, SVM	Sieci neuronowe

Wykorzystanie języka Python w „Uczeniu maszynowym”

- Dziedzina sztucznej inteligencji (AI)
- Algorytmy uczą się na podstawie danych
- Główne zastosowania:
- Rozpoznawanie obrazów i głosu
- Analiza tekstu i predykcja
- Systemy rekomendacji (np. Netflix, YouTube)
- Samojezdne pojazdy, analiza medyczna



Popularne biblioteki



Scikit-learn – klasyfikacja, regresja, klasteryzacja

TensorFlow – sieci neuronowe, deep learning (Google)

Keras – prosty interfejs do TensorFlow

PyTorch – alternatywa dla TensorFlow (Meta)

Pandas, NumPy – analiza danych i operacje matematyczne

Matplotlib, Seaborn – wizualizacja danych

Przykład zastosowania – drzewo decyzyjne

```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
iris = load_iris()
X, y = iris.data, iris.target
model = DecisionTreeClassifier()
model.fit(X, y)
print(model.predict([[5.1, 3.5, 1.4, 0.2]]))
```

Drzewo decyzyjne. Wybieramy gatunek na podstawie cechy. Wypisany w konsoli indeks będzie wybranym kwiatem.

Sieć neuronowa

- **Zastosowanie:** klasyfikacja, regresja
- **Opis:** warstwy neuronów połączone wagami; uczą się przez propagację błędu.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential([
   Dense(64, activation='relu',
input_shape=(X.shape[1],)),
   Dense(1, activation='sigmoid')
])
```

Co można zrobić?

Import danych – z pliku CSV, bazy danych, API

Analiza i czyszczenie danych – Pandas, NumPy

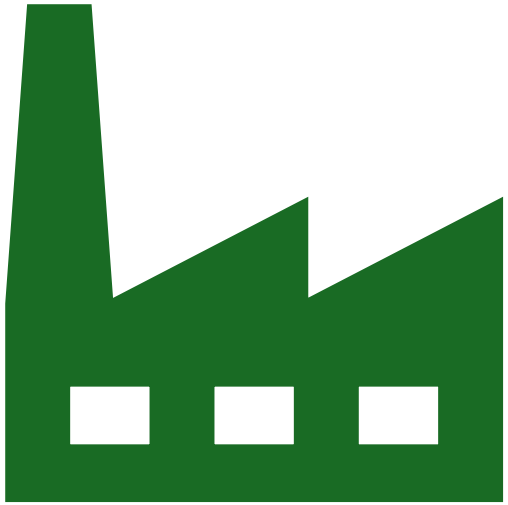
Wybór modelu ML – np. regresja liniowa, drzewa decyzyjne

Trenowanie modelu – Scikit-learn, TensorFlow

Ewaluacja modelu – np. accuracy, precision, recall

Zastosowanie modelu – predykcja nowych danych

Przykładowy kod – Tensor i Kieras



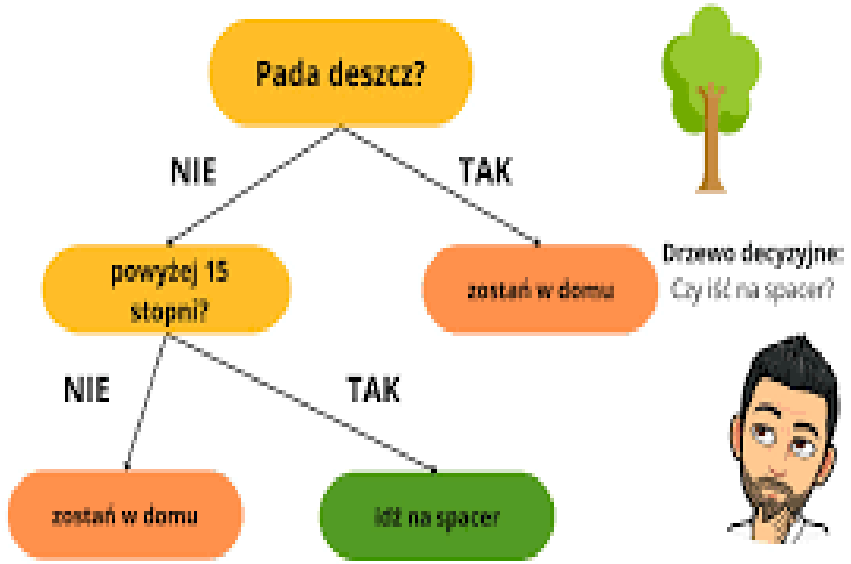
```
import gym
import numpy as np
import random
from collections import deque
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam

# Parametry środowiska
env = gym.make('CartPole-v1')
state_size = env.observation_space.shape[0]
action_size = env.action_space.n

# Parametry agenta
gamma = 0.95    # Współczynnik dyskontowania
epsilon = 1.0   # Początkowy poziom eksploracji
epsilon_min = 0.01
epsilon_decay = 0.995
learning_rate = 0.001
batch_size = 64
memory = deque(maxlen=2000)

# Budujemy model sieci neuronowej
def build_model():
    model = Sequential()
    model.add(Dense(24, input_dim=state_size, activation='relu'))
    model.add(Dense(24, activation='relu'))
    model.add(Dense(action_size, activation='linear'))
    model.compile(loss='mse', optimizer=Adam(learning_rate=learning_rate))
    return model

model = build_model()

# Funkcja zapamiętywania doświadczeń
def remember(state, action, reward, next_state, done):
    memory.append((state, action, reward, next_state, done))

# Funkcja eksploracji/eksploatacji
def act(state):
    if np.random.rand() <= epsilon:
        return random.randrange(action_size)
    q_values = model.predict(state, verbose=0)
    return np.argmax(q_values[0])

# Trening z próbek pamięci
def replay():
    global epsilon
    if len(memory) < batch_size:
        return
    minibatch = random.sample(memory, batch_size)
    for state, action, reward, next_state, done in minibatch:
        target = reward
        if not done:
            target += gamma * np.amax(model.predict(next_state, verbose=0)[0])
        target_f = model.predict(state, verbose=0)
        target_f[0][action] = target
        model.fit(state, target_f, epochs=1, verbose=0)
    if epsilon > epsilon_min:
        epsilon *= epsilon_decay

# Trening agenta
episodes = 500

for e in range(episodes):
    state = env.reset()
    state = np.reshape(state, [1, state_size])
    for time in range(500):
        # env.render()  # Odkomentuj, by widzieć animację
        action = act(state)
        next_state, reward, done, _ = env.step(action)
        reward = reward if not done else -10
        next_state = np.reshape(next_state, [1, state_size])
        remember(state, action, reward, next_state, done)
        state = next_state
        if done:
            print(f"Epizod: {e + 1}/{episodes}, wynik: {time}, epsilon: {epsilon:.2f}")
            break
    replay()
```

Przykład zastosowania – klasyfikacja SMS

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB

# Dane treningowe
sms = ["Darmowe wakacje!", "Spotkajmy się jutro", "Wygraj
iPhone’a!", "Cześć, co u ciebie?"]
labels = [1, 0, 1, 0]  # 1 = spam, 0 = nie-spam

# Przekształcenie tekstu na wektor liczb
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(sms)

# Model Naive Bayes
model = MultinomialNB()
model.fit(X, labels)

# Przewidywanie
nowy_sms = vectorizer.transform(["Wygraj nagrodę już dziś!"])
print("Spam?" , model.predict(nowy_sms)[0])
```

Program sprawdza czy dany SMS to spam czy nie.

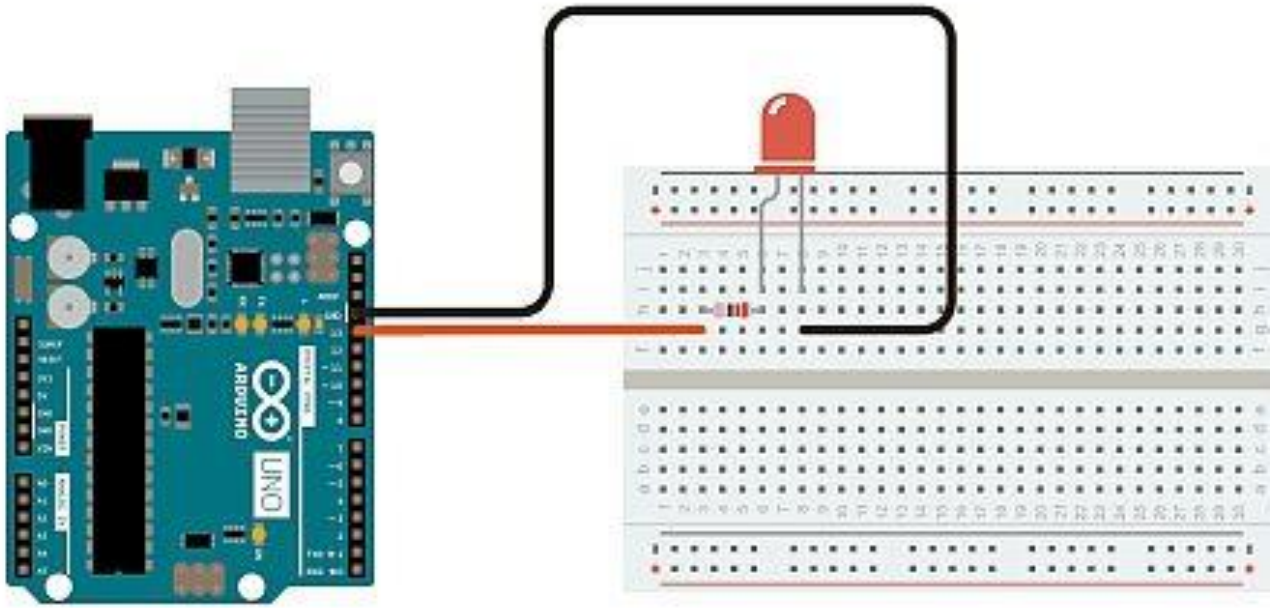
Przykład zastosowania – przewidywanie ceny




```
import tkinter as tk
from sklearn.linear_model import
LinearRegression
import numpy as np

# Dane treningowe (przykładowe)
# [rok, przebieg w km, moc w KM]
X = np.array([
   [2015, 120000, 90],
   [2018, 80000, 110],
   [2020, 50000, 130],
   [2012, 150000, 75],
   [2019, 60000, 115]
])
y = np.array([25000, 40000, 55000, 18000,
45000])  # ceny w PLN

# Trenowanie modelu
model = LinearRegression()
model.fit(X, y)

# Funkcja predykcji
def przewiduj_cene():
   try:
      rok = int(entry_rok.get())
      przebieg = int(entry_przebieg.get())
      moc = int(entry_moc.get())
      dane = np.array([[rok, przebieg, moc]])
      cena = model.predict(dane)[0]
      wynik_label.config(text=f"Szacowana
cena: {int(cena):,} PLN")
   except ValueError:
      wynik_label.config(text="Błąd danych
wejściowych.")

# GUI – Tkinter
root = tk.Tk()
root.title("Przewidywanie ceny samochodu")

tk.Label(root, text="Rok
produkcji").grid(row=0, column=0)
entry_rok = tk.Entry(root)
entry_rok.grid(row=0, column=1)

tk.Label(root, text="Przebieg
(km)").grid(row=1, column=0)
entry_przebieg = tk.Entry(root)
entry_przebieg.grid(row=1, column=1)

tk.Label(root, text="Moc (KM)").grid(row=2,
column=0)
entry_moc = tk.Entry(root)
entry_moc.grid(row=2, column=1)

tk.Button(root, text="Przewiduj cenę",
command=przewiduj_cene).grid(row=3,
column=0, columnspan=2, pady=10)

wynik_label = tk.Label(root, text="",
fg="blue")
wynik_label.grid(row=4, column=0,
columnspan=2)

root.mainloop()
```

Wykorzystanie Python do obsługi kontrolerów

- silniki (DC, krokowe, serwo),
- czujniki (np. temperatury, odległości),
- przyciski, LED-y, wyświetlacze,
- joysticki, pady, klawiatury sterujące,
- Arduino przez port szeregowy (USB),
- Raspberry Pi przez piny GPIO.




```
import RPi.GPIO as GPIO
import time# Ustawienie trybu numeracji pinów
GPIO.setmode(GPIO.BCM)# Konfiguracja pinów
led_pin = 18
motor_pin = 23
GPIO.setup(led_pin, GPIO.OUT)
GPIO.setup(motor_pin, GPIO.OUT)
try:
while True:
        print("Włączam LED i silnik")
        GPIO.output(led_pin, GPIO.HIGH)
        GPIO.output(motor_pin, GPIO.HIGH)
        time.sleep(2)
        print("Wyłączam LED i silnik")
        GPIO.output(led_pin, GPIO.LOW)
        GPIO.output(motor_pin, GPIO.LOW)
 time.sleep(2)
except KeyboardInterrupt:
 GPIO.cleanup()
```

Przykład – GPIO, Arduino i LED

Przykład – obsługa joystick w grze

```
import pygame

pygame.init()

pygame.joystick.init()# Sprawdzenie liczby podłączonych joysticków

joystick_count = pygame.joystick.get_count()

print(f"Znaleziono {joystick_count} joysticków.")

if joystick_count == 0:

        print("Brak joysticka!")

else:

        joystick = pygame.joystick.Joystick(0)

joystick.init()

while True:

        pygame.event.pump()

        x = joystick.get_axis(0)

        y = joystick.get_axis(1)

print(f"Pozycja X: {x:.2f}, Y: {y:.2f}")
```

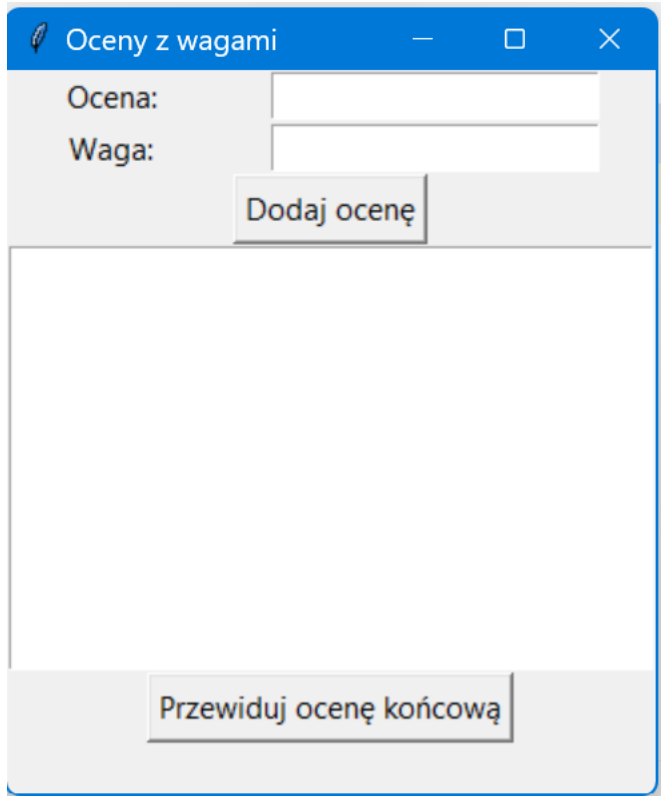
Co można kontrolować za pomocą Pythona?

Urządzenie	Biblioteka Python	Interfejs
Arduino	pyserial	USB / COM
Raspberry Pi	RPi.GPIO, gpiozero	GPIO
Joystick/Gamepad	pygame, inputs	USB / Bluetooth
Robot (np. LEGO EV3)	ev3dev2, pybricks	Bluetooth / USB
CNC / drukarki 3D	pyserial, G-code	COM / G-code

Zadania do samodzielnego wykonania

Zadanie 1

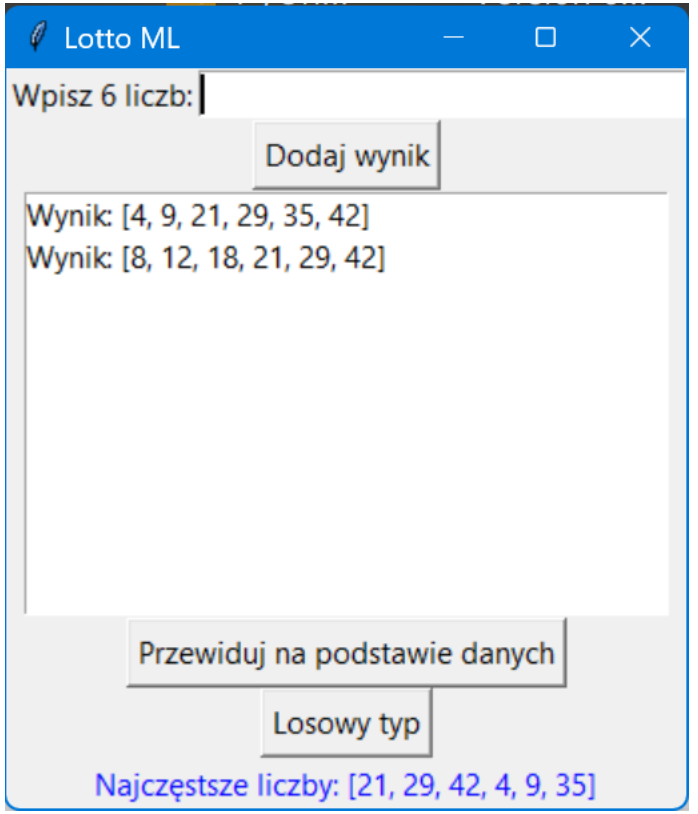
Aplikacja przewiduje ocenę końcową na podstawie ocen i ich wag.



Zadania do samodzielnego wykonania

Zadanie 2

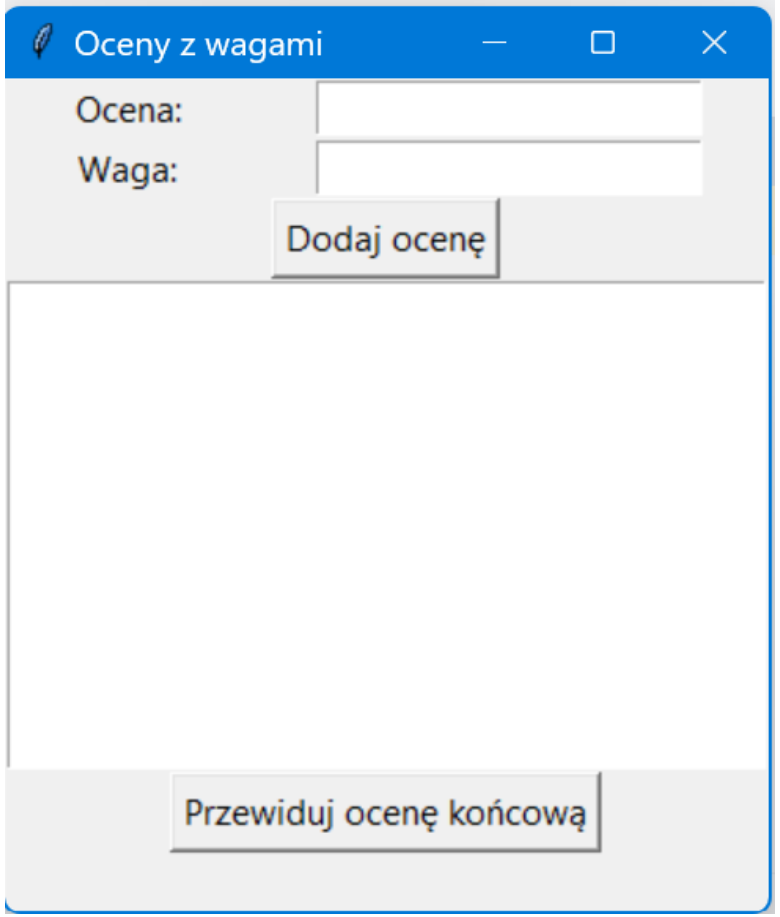
Aplikacja pobiera od użytkownika liczby, następnie typuje przewidywane na podstawie ilości powtórzeń oraz generuje losowy zestaw.



Zadania do samodzielnego wykonania

Zadanie 3

Aplikacja pobiera od użytkownika dane. Dane są przechowywane w tablicach. Zastosowano regresję liniową oraz przedstawienie danych na wykresie.



Python na egzaminie INF.04

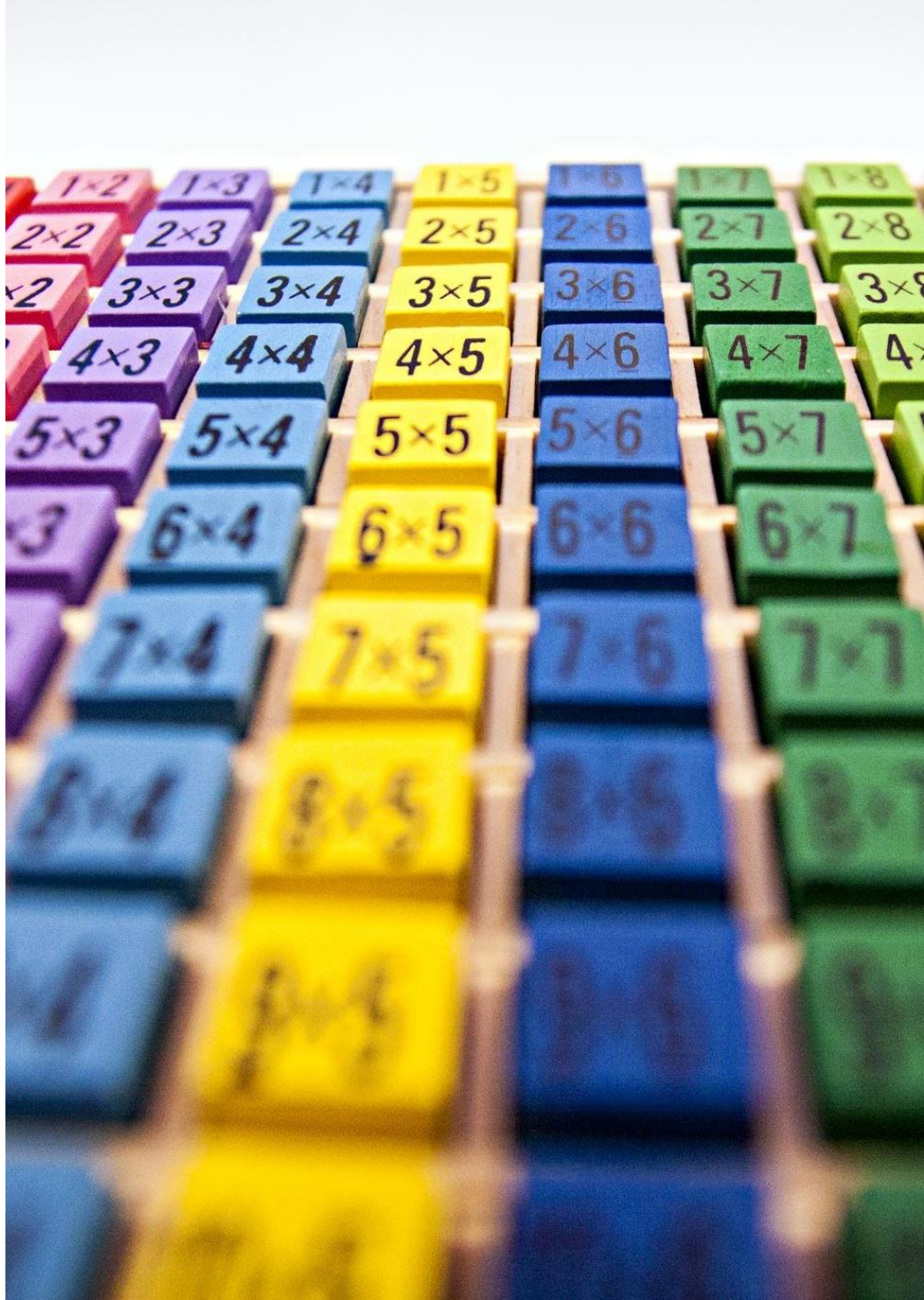
Język Python na egzaminie może być używany do aplikacji konsolowej lub desktopowej.

Aplikacja konsolowa to zwykle:

- Testy jednostkowe aplikacji;
- Wybrany rodzaj sortowania.

Aplikację desktopową najczęściej stanowi:

- Okno, w którym użytkownik wpisuje dane, następnie zostają posortowane, zapisane lub wyszukane;
- Aplikacja w formie prostej gry lub odtwarzacza plików dźwiękowych, tekstowych lub tabeli z możliwością wybierania danych.



Testy jednostkowe

```
def dodaj(a, b):
    return a + b
```

Narzędzia do testów:

- unittest (wbudowany w Pythona)
- pytest (bardziej zaawansowany)
- mock – do tworzenia atrap np. dla playera

```
import unittest

class TestDodaj(unittest.TestCase):
    def test_dodaj(self):
        self.assertEqual(dodaj(2, 3), 5)
        self.assertEqual(dodaj(-1, 1), 0)
        self.assertEqual(dodaj(0, 0), 0)

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

Testy jednostkowe

Zalety testów jednostkowych:

- szybkie wykrywanie błędów
- łatwe testowanie zmian bez uruchamiania całej aplikacji
- większa pewność, że aplikacja działa poprawnie
- pomagają przy refaktoryzacji

Co testujemy?	Jakie testy?
Funkcja dodawania pliku do playlisty	Czy plik trafia na listę?
Sortowanie playlisty	Czy lista jest poprawnie posortowana?
Regulacja głośności	Czy set_volume(50) ustawia 50% w VLC?
Obsługa błędów	Czy brak pliku zgłasza komunikat?

Algorytmy sortowania

```
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        # Ostatnie i elementów jest już na właściwym miejscu
        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
                # Zamiana miejscami, jeśli element jest większy od następnego
                arr[j], arr[j + 1] = arr[j + 1], arr[j]

# Przykład użycia:
lista = [5, 2, 9, 1, 5, 6]
bubble_sort(lista)
print(lista)  # Wynik: [1, 2, 5, 5, 6, 9]
```

Obok przykład sortowania bąbelkowego. Zadania dotyczą również innych algorytmów sortowania: kubełkowego, przez wstawianie, tzw. Quick Sort. Może być również „Szyfr Cezara” lub algorytm Euklidesa.

Czasem są to proste zadania wykorzystujące tablice oraz zapis i odczyt z pliku.

Algorytmy sortowania

```
def szyfr_cezara(tekst, przesuniecie):
    wynik = ""
    for znak in tekst:
        if znak.isalpha():
            # Ustal bazę (A lub a) w zależności od wielkości litery
            baza = ord('A') if znak.isupper() else ord('a')
            # Przesuń literę w alfabecie z zawijaniem (mod 26)
            nowy_znak = chr((ord(znak) - baza + przesuniecie) % 26 + baza)
            wynik += nowy_znak
        else:
            # Pozostaw inne znaki bez zmian
            wynik += znak
    return wynik

# Przykład użycia:
tekst = "Ala ma kota!"
przesuniecie = 3
zaszyfrowany = szyfr_cezara(tekst, przesuniecie)
print(zaszyfrowany)  # Wynik: "Dod pd nrwd!"
```

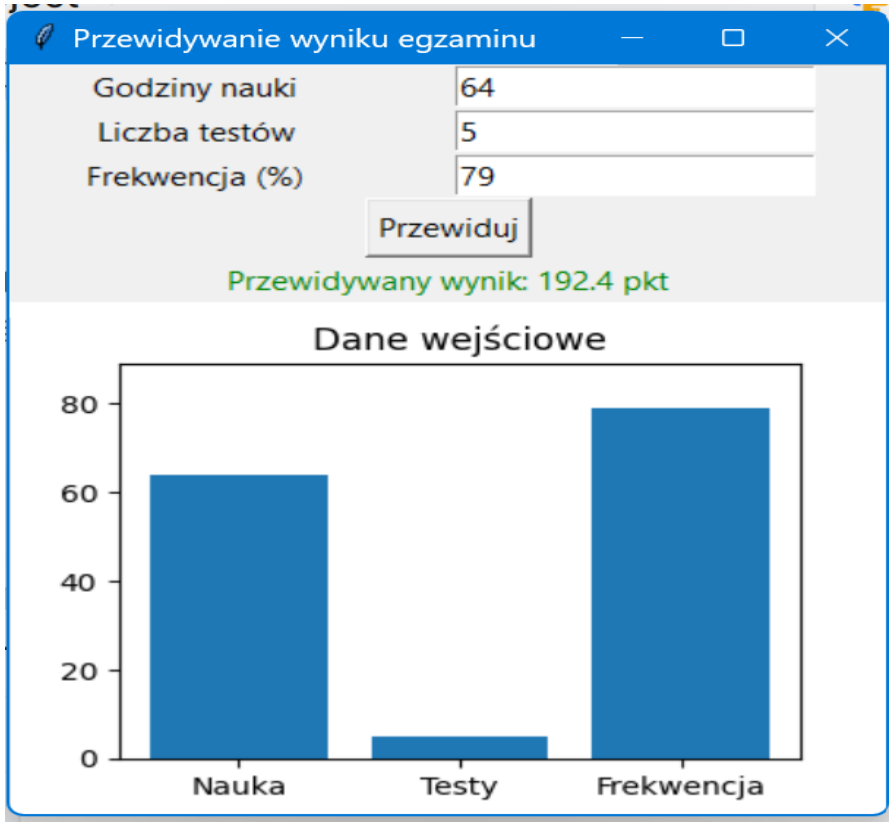
Szyfr Cezara, którego zadaniem jest zakodowanie ciągu znaków w taki sposób, że każda litera może być zastąpiona literą następującą po sobie, np. z litery A zrobimy D dodając zmienną, której nadamy wartość 3.

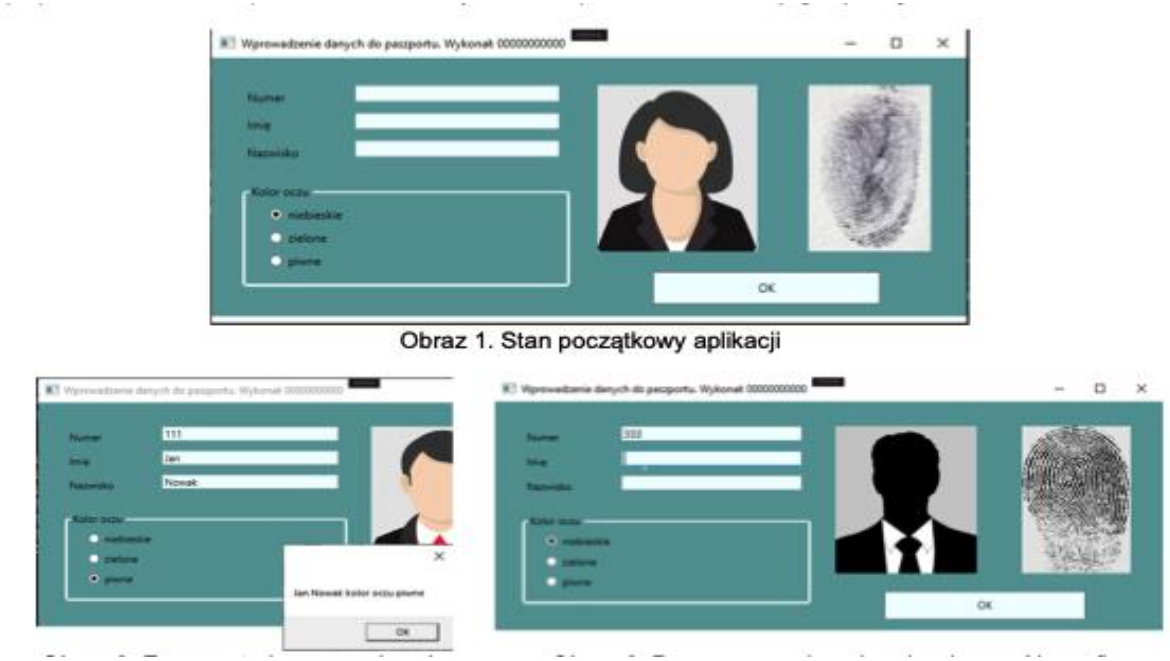
Program deszyfrujący najczęściej zawiera jedynie przeciwną wartość zmiennej, tj. -3.

Typowe zadanie egzaminacyjne

Okno z wyszukiwarką i „RANDOM na obiektach” czyli możliwość wyszukania danych. Zadanie może dotyczyć również możliwości przechowywania i odczytu danych w pliku JSON lub w bazie SQLite.

Arkusz zwykle zawiera zadanie w formie sporządzenia dokumentacji po wykonaniu wcześniejszych zadań. Może być to forma komentarzy, tabeli z wpisanymi zmiennymi, funkcjami i klasami lub w formie dokumentu skróconej instrukcji obsługi.





Obraz 1. Stan początkowy aplikacji

Typowe zadanie egzaminacyjne

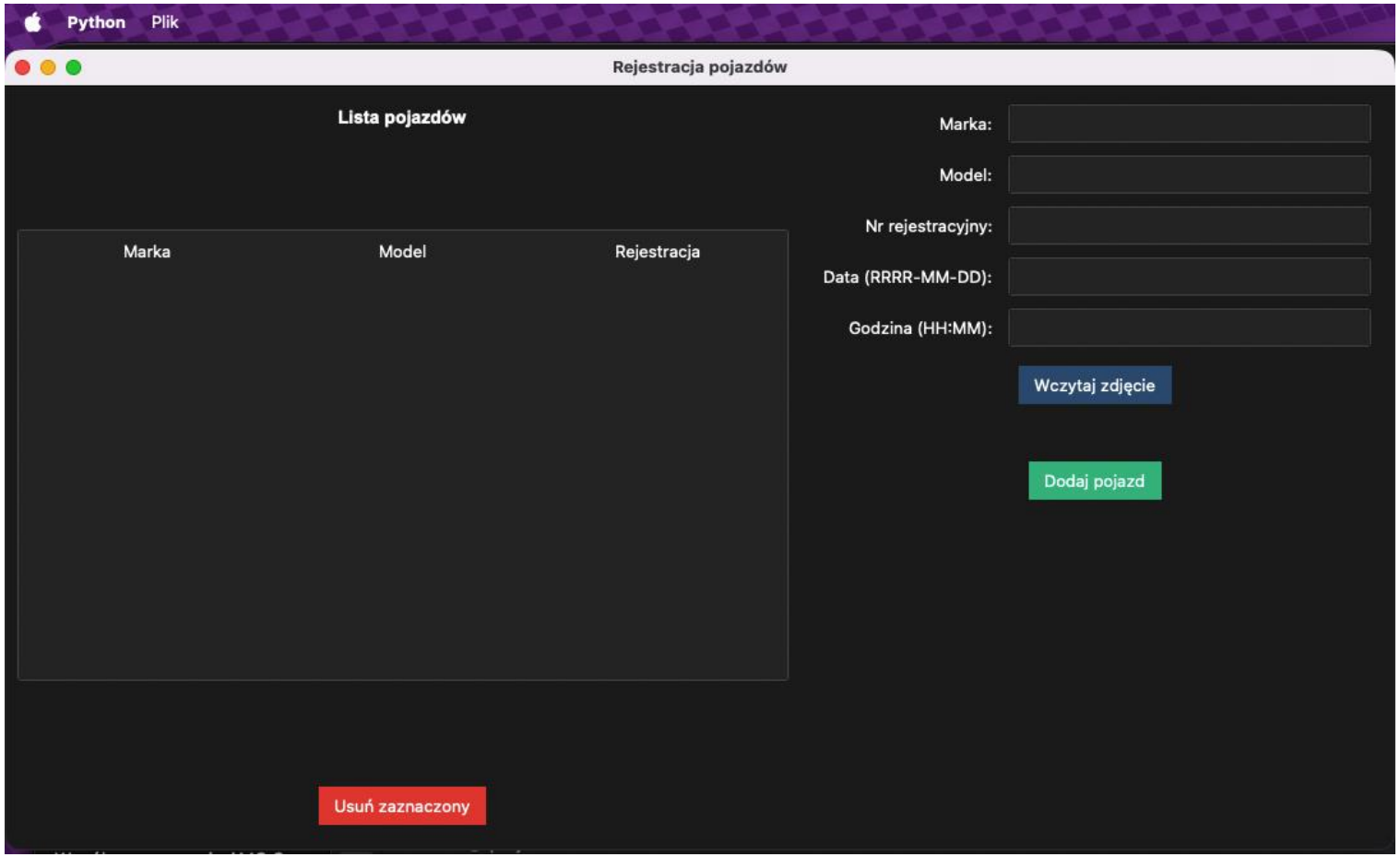
Typowe zadanie egzaminacyjne

Obraz 1. Stan początkowy aplikacji

Obraz 2. Fragment okna po wybraniu przycisku „Generuj hasło"

Obraz 3. Po wybraniu przycisku „Zatwierdź"

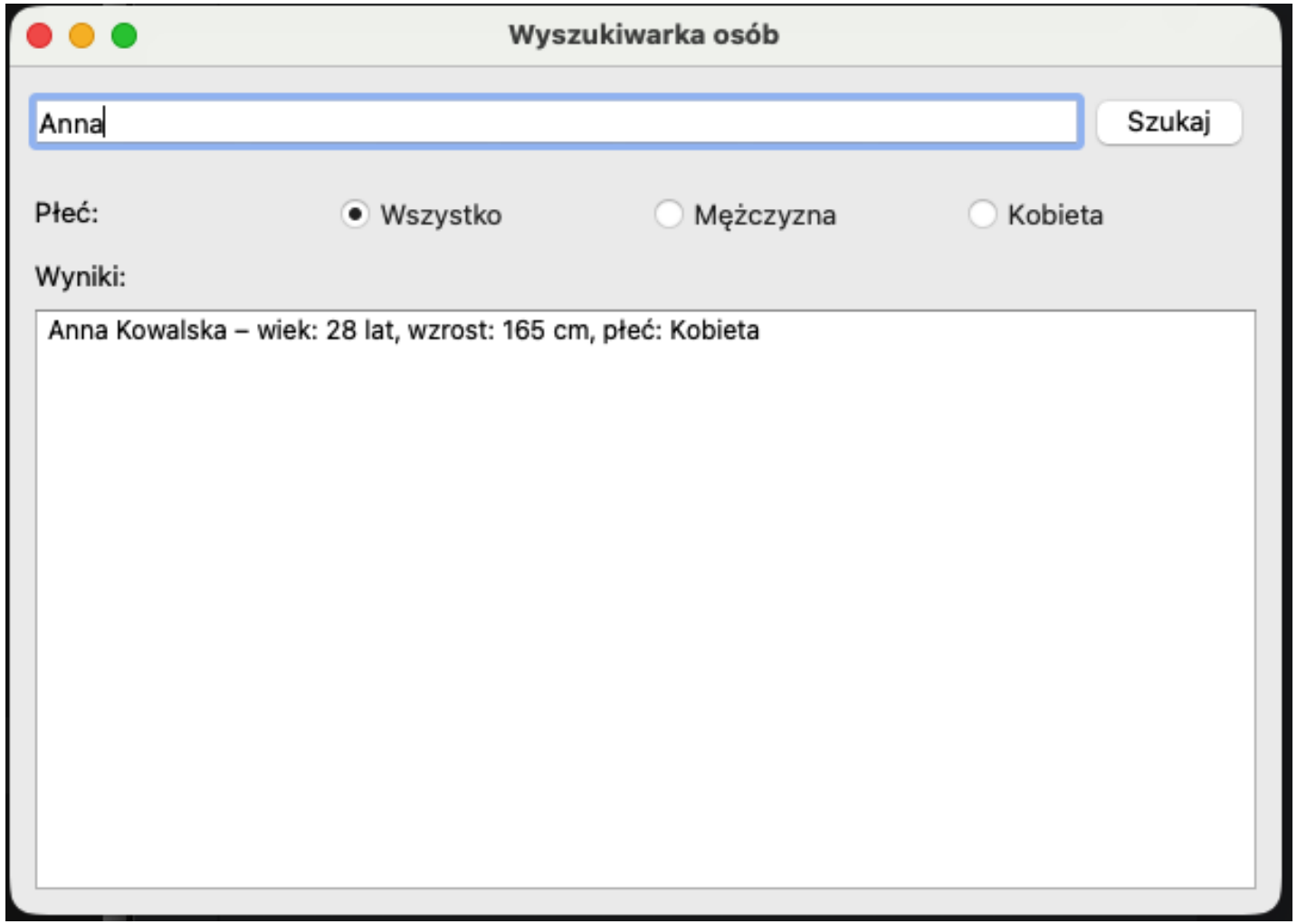
Typowe zadanie



Kod do wykorzystania

```
import tkinter as tk
from ttkbootstrap import Style
from ttkbootstrap.widgets import Entry, Button, Label, Frame, Treeview
from tkinter import filedialog, messagebox
from PIL import Image, ImageTk
import csv
import os
import re
from datetime import datetime

class VehicleApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Rejestracja pojazdów")
        self.root.geometry("1000x600")

        self.style = Style("darkly") # lub: flatly, darkly, morph, etc.
        self.vehicles = []
        self.photo = None
        self.image_path = None

        # Layout
        self.left_frame = Frame(root, padding=10)
        self.left_frame.pack(side=tk.LEFT, fill=tk.Y)

        self.right_frame = Frame(root, padding=10)
        self.right_frame.pack(side=tk.RIGHT, expand=True, fill=tk.BOTH)

        self.create_list()
        self.create_form()
        self.create_menu()
        self.load_data_from_file()

    def create_menu(self):
        menubar = tk.Menu(self.root)
        file_menu = tk.Menu(menubar, tearoff=0)
        file_menu.add_command(label="Zamknij",
command=self.root.quit)
        menubar.add_cascade(label="Plik", menu=file_menu)
        self.root.config(menu=menubar)

    def create_list(self):
        Label(self.left_frame, text="Lista pojazdów", font=("Arial", 14,
"bold")).pack(pady=5)

        self.tree = Treeview(self.left_frame, columns=("Marka", "Model",
"Rejestracja"), show="headings", height=20)
        for col in self.tree["columns"]:
            self.tree.heading(col, text=col)
        self.tree.pack(pady=5, expand=True)

        Button(self.left_frame, text="Usuń zaznaczony", bootstyle="danger",
command=self.delete_selected).pack(pady=10)

    def create_form(self):
        labels = ["Marka:", "Model:", "Nr rejestracyjny:", "Data (RRRR-MM-
DD):", "Godzina (HH:MM):"]
        self.entries = {}

        for i, label in enumerate(labels):
            Label(self.right_frame, text=label).grid(row=i, column=0,
sticky="e", padx=5, pady=5)
            entry = Entry(self.right_frame, width=30)
            entry.grid(row=i, column=1, sticky="w", padx=5, pady=5)
            self.entries[label] = entry

        Button(self.right_frame, text="Wczytaj zdjęcie",
command=self.load_image).grid(row=5, column=0, columnspan=2,
pady=10)

        self.image_label = Label(self.right_frame)
        self.image_label.grid(row=6, column=0, columnspan=2)

        Button(self.right_frame, text="Dodaj pojazd", bootstyle="success",
command=self.validate_and_add).grid(row=7, column=0,
columnspan=2, pady=15)

    def load_image(self):
        path = filedialog.askopenfilename(filetypes=[("Obrazy", "*.png *.jpg
*.jpeg *.gif")])
        if path:
            img = Image.open(path)
            img = img.resize((200, 150))
            self.photo = ImageTk.PhotoImage(img)
            self.image_label.config(image=self.photo)
            self.image_label.image = self.photo
            self.image_path = path

    def validate_and_add(self):
        marka = self.entries["Marka:"].get().strip()
        model = self.entries["Model:"].get().strip()
        nr = self.entries["Nr rejestracyjny:"].get().strip()
        data = self.entries["Data (RRRR-MM-DD):"].get().strip()
        godzina = self.entries["Godzina (HH:MM):"].get().strip()

        if not marka or not model or not nr or not data or not godzina:
            messagebox.showerror("Błąd", "Wypełnij wszystkie pola.")
            return

        if not re.match(r"^[A-Z]{1,3}?[0-9A-Z]{3,5}$", nr):
            messagebox.showerror("Błąd", "Nieprawidłowy numer
rejestracyjny.")
            return

        try:
            datetime.strptime(data, "%Y-%m-%d")
            datetime.strptime(godzina, "%H:%M")
        except ValueError:
            messagebox.showerror("Błąd", "Nieprawidłowy format daty lub
godziny.")
            return

        # Dodaj pojazd do listy i pliku
        self.vehicles.append((marka, model, nr, data, godzina,
self.image_path))
        self.tree.insert("", tk.END, values=(marka, model, nr))
        self.save_to_file()
        self.clear_form()
        messagebox.showinfo("Sukces", "Pojazd dodany.")

    def delete_selected(self):
        selected = self.tree.selection()
        if not selected:
            messagebox.showwarning("Uwaga", "Zaznacz pojazd do
usunięcia.")
            return

        for item in selected:
            values = self.tree.item(item, "values")
            marka, model, nr = values[0], values[1], values[2]

            self.tree.delete(item)

            self.vehicles = [v for v in self.vehicles if not (v[0] == marka and v[1]
==model and v[2] == nr)]

        self.save_to_file()
        messagebox.showinfo("Usunięto", "Zaznaczony pojazd został
usunięty.")

    def clear_form(self):
        for entry in self.entries.values():
            entry.delete(0, tk.END)
        self.image_label.config(image="")
        self.photo = None
        self.image_path = None

    def save_to_file(self):
        with open("pojazdy.csv", "w", newline="", encoding="utf-8") as f:
            writer = csv.writer(f)
            writer.writerow(["marka", "model", "nr_rejestracyjny", "data",
"godzina", "zdjecie"])
            for row in self.vehicles:
                writer.writerow(row)

    def load_data_from_file(self):
        if os.path.exists("pojazdy.csv"):
            with open("pojazdy.csv", "r", newline="", encoding="utf-8") as f:
                reader = csv.DictReader(f)
                for row in reader:
                    self.vehicles.append((row["marka"], row["model"],
row["nr_rejestracyjny"], row["data"], row["godzina"], row["zdjecie"]))
                    self.tree.insert("", tk.END, values=(row["marka"], row["model"],
row["nr_rejestracyjny"]))

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

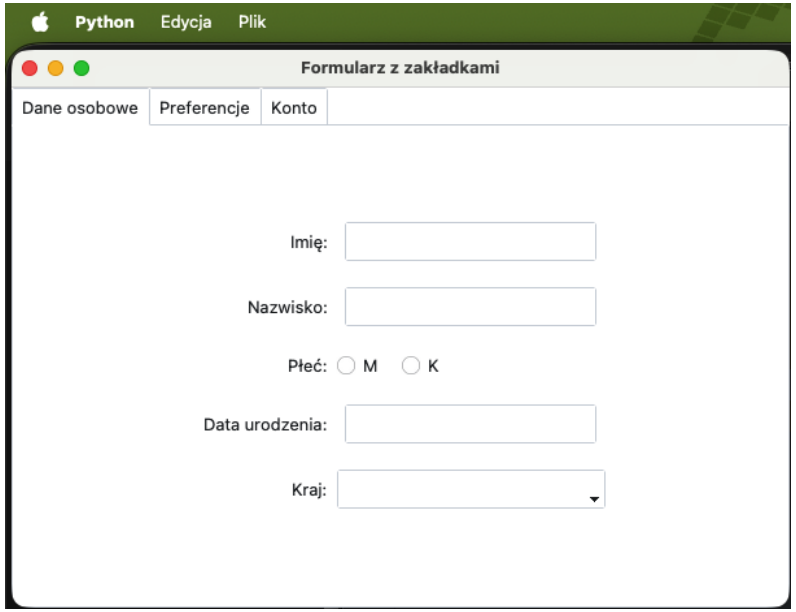
Wyszukiwarka danych – „pewniak”

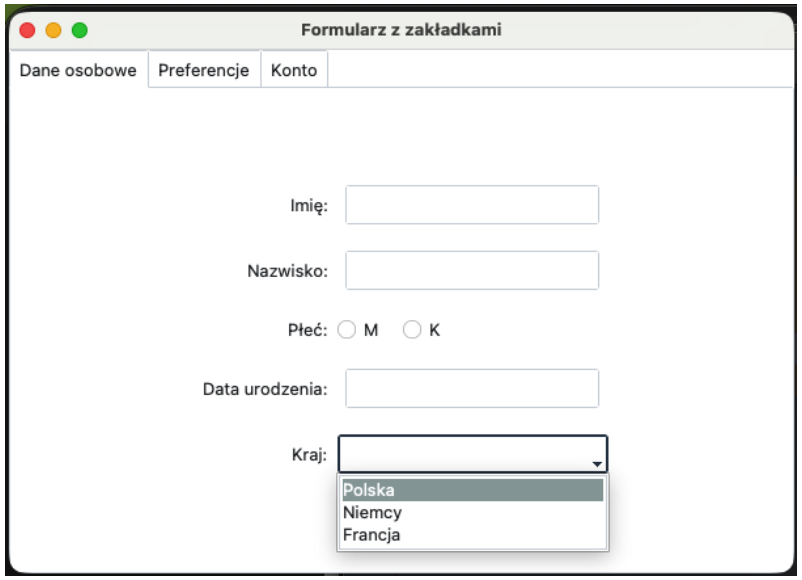


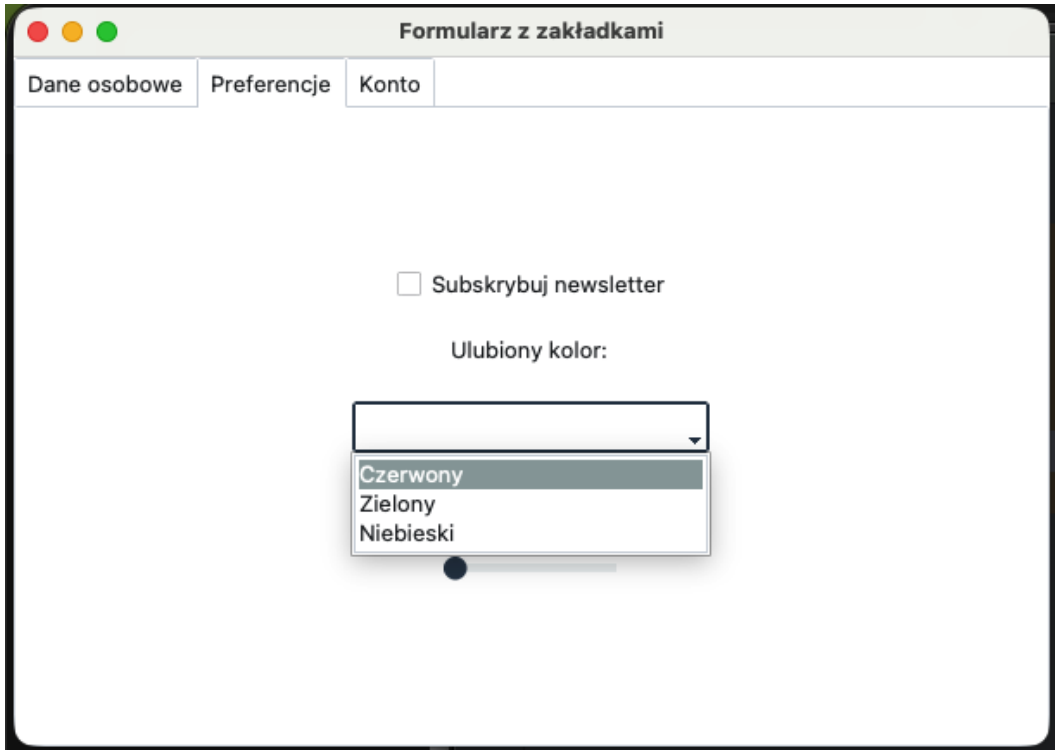
Kod programu

```
import sys
from PyQt5.QtWidgets import (
    QApplication, QMainWindow, QWidget, QVBoxLayout,
    QHBoxLayout, QLineEdit, QPushButton, QListWidget,
    QLabel, QRadioButton, QButtonGroup
)

class SearchApp(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Wyszukiwarka osób")
        self.setGeometry(100, 100, 600, 400)

        # Dane osób – teraz z płcią
        self.people = [
            {"imie": "Anna", "nazwisko": "Kowalska", "wiek": 28, "wzrost": 165, "plec": "K"},
            {"imie": "Jan", "nazwisko": "Nowak", "wiek": 34, "wzrost": 180, "plec": "M"},
            {"imie": "Piotr", "nazwisko": "Wiśniewski", "wiek": 22, "wzrost": 175, "plec": "M"},
            {"imie": "Maria", "nazwisko": "Zielińska", "wiek": 30, "wzrost": 160, "plec": "K"},
            {"imie": "Tomasz", "nazwisko": "Kowalczyk", "wiek": 27, "wzrost": 185, "plec": "M"},
        ]

        # Widgety
        self.input = QLineEdit()
        self.input.setPlaceholderText("Wpisz imię, nazwisko, wiek lub wzrost...")

        self.search_button = QPushButton("Szukaj")
        self.result_list = QListWidget()

        # Radio przyciski dla płci
        self.radio_all = QRadioButton("Wszystko")
        self.radio_male = QRadioButton("Mężczyzna")
        self.radio_female = QRadioButton("Kobieta")
        self.radio_all.setChecked(True)

        # Grupa przycisków (wymusza wybór jednego)
        self.gender_group = QButtonGroup()
        self.gender_group.addButton(self.radio_all)
        self.gender_group.addButton(self.radio_male)
        self.gender_group.addButton(self.radio_female)

        # Layouty
        gender_layout = QHBoxLayout()
        gender_layout.addWidget(QLabel("Płeć:"))
        gender_layout.addWidget(self.radio_all)
        gender_layout.addWidget(self.radio_male)
        gender_layout.addWidget(self.radio_female)

        top_layout = QHBoxLayout()
        top_layout.addWidget(self.input)
        top_layout.addWidget(self.search_button)

        main_layout = QVBoxLayout()
        main_layout.addLayout(top_layout)
        main_layout.addLayout(gender_layout)
        main_layout.addWidget(QLabel("Wyniki:"))
        main_layout.addWidget(self.result_list)

        container = QWidget()
        container.setLayout(main_layout)
        self.setCentralWidget(container)

        # Połączenia
        self.search_button.clicked.connect(self.search_people)

    def search_people(self):
        query = self.input.text().strip().lower()
        selected_gender = None
        if self.radio_male.isChecked():
            selected_gender = "M"
        elif self.radio_female.isChecked():
            selected_gender = "K"

        self.result_list.clear()

        results = []
        for person in self.people:
            matches_query = (
                query in person["imie"].lower()
                or query in person["nazwisko"].lower()
                or query in str(person["wiek"])
                or query in str(person["wzrost"])
            )

            matches_gender = selected_gender is None or person["plec"] == selected_gender

            if matches_query and matches_gender:
                text = f"{person['imie']} {person['nazwisko']} – wiek: {person['wiek']} lat, wzrost:
{person['wzrost']} cm, płeć: {'Mężczyzna' if person['plec'] == 'M' else 'Kobieta'}"
                results.append(text)

        if results:
            self.result_list.addItems(results)
        else:
            self.result_list.addItem("Brak wyników.")

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

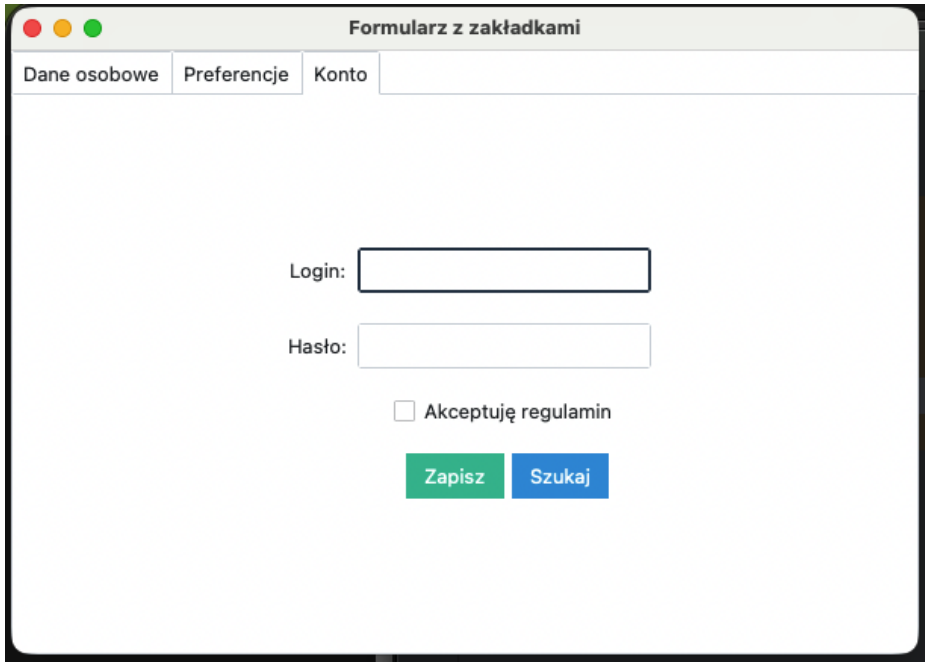
Formularz, karty, menu i zapis

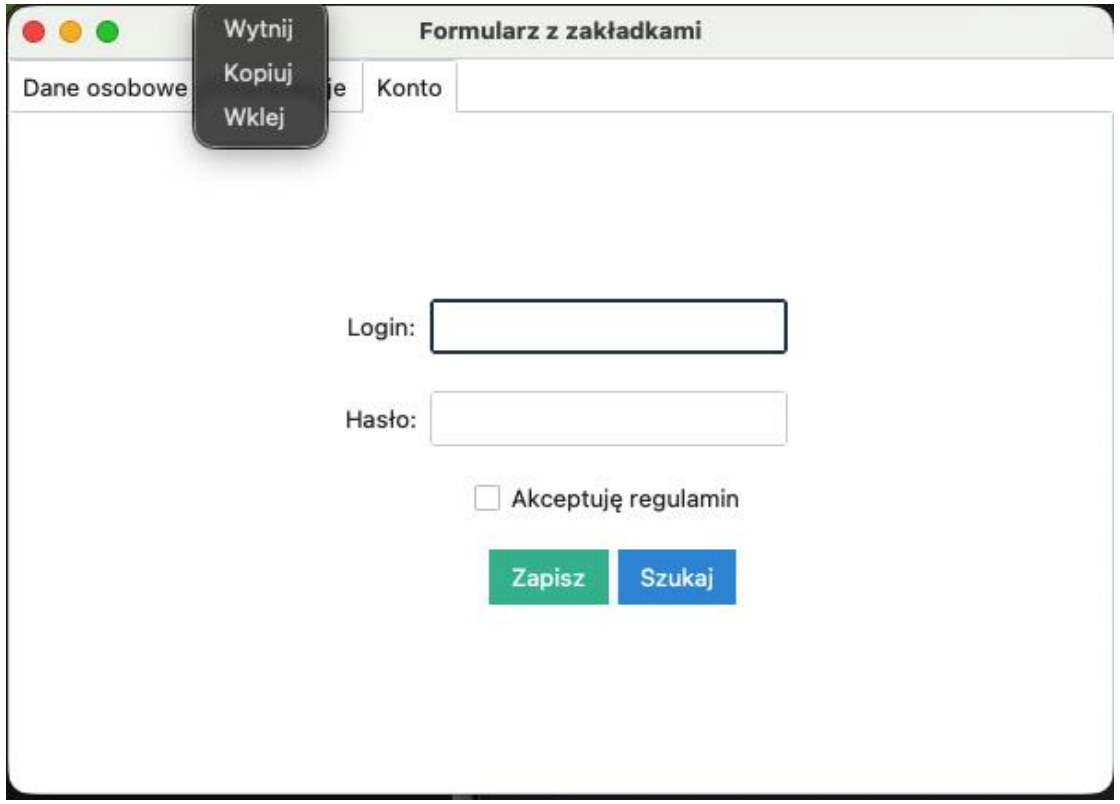






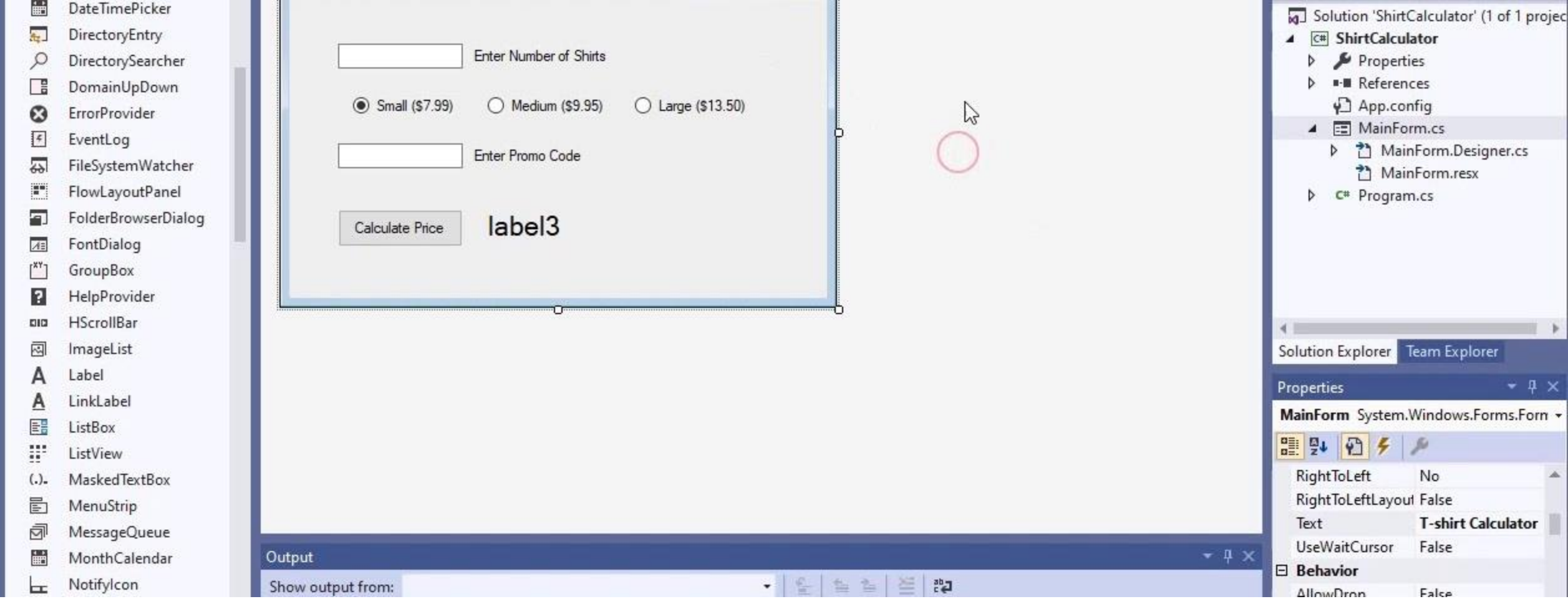
Formularz, karty, menu i zapis





Kod programu

```
import ttkbootstrap as ttkb
from ttkbootstrap.constants import *
import tkinter as tk
from tkinter import messagebox
import json
import os

plik = "dane.json"

def zapisz():
    dane = {k: v.get() for k, v in entries.items()}
    dane["płeć"] = plec.get()

    if not dane["imię"] or not dane["nazwisko"]:
        messagebox.showwarning("Błąd", "Wprowadź imię i nazwisko.")
        return

    lista = []
    if os.path.exists(plik):
        with open(plik, "r", encoding="utf-8") as f:
            try:
                lista = json.load(f)
            except json.JSONDecodeError:
                lista = []

    lista.append(dane)
    with open(plik, "w", encoding="utf-8") as f:
        json.dump(lista, f, ensure_ascii=False, indent=2)
    messagebox.showinfo("Sukces", "Dane zapisane!")

def szukaj():
    kryterium = entries["imię"].get().lower()
    nazwisko = entries["nazwisko"].get().lower()

    if not kryterium and not nazwisko:
        messagebox.showinfo("Wyszukiwanie", "Wprowadź imię lub nazwisko.")
        return

    if not os.path.exists(plik):
        messagebox.showwarning("Brak danych", "Brak zapisanych danych.")
        return

    with open(plik, "r", encoding="utf-8") as f:
        lista = json.load(f)

    znalezione = [
        d for d in lista
        if kryterium in d["imię"].lower() or nazwisko in d["nazwisko"].lower()
    ]

    if znalezione:
        wynik = "\n\n".join([f"{d['imię']} {d['nazwisko']} - {d['login']}" for d in znalezione])
        messagebox.showinfo("Wyniki wyszukiwania", wynik)
    else:
        messagebox.showinfo("Brak wyników", "Nie znaleziono pasujących danych.")

def wytnij():
    widget = root.focus_get()
    if isinstance(widget, (tk.Entry, ttkb.Entry)):
        widget.event_generate("<<Cut>>")

def kopiuj():
    widget = root.focus_get()
    if isinstance(widget, (tk.Entry, ttkb.Entry)):
        widget.event_generate("<<Copy>>")

def wklej():
    widget = root.focus_get()
    if isinstance(widget, (tk.Entry, ttkb.Entry)):
        widget.event_generate("<<Paste>>")

root = ttkb.Window(themename="flatly")
root.title("Formularz z zakładkami")
root.geometry("600x400")

# MENU
menu_bar = tk.Menu(root)
root.config(menu=menu_bar)

# Menu Edycja
edit_menu = tk.Menu(menu_bar, tearoff=0)
edit_menu.add_command(label="Wytnij", command=wytnij)
edit_menu.add_command(label="Kopiuj", command=kopiuj)
edit_menu.add_command(label="Wklej", command=wklej)
menu_bar.add_cascade(label="Edycja", menu=edit_menu)

# Menu Plik
file_menu = tk.Menu(menu_bar, tearoff=0)
file_menu.add_command(label="Zapisz", command=zapisz)
menu_bar.add_cascade(label="Plik", menu=file_menu)

notebook = ttkb.Notebook(root)
notebook.pack(fill='both', expand=True)

entries = {
    "imię": tk.StringVar(),
    "nazwisko": tk.StringVar(),
    "data": tk.StringVar(),
    "kraj": tk.StringVar(),
    "newsletter": tk.BooleanVar(),
    "kolor": tk.StringVar(),
    "głośność": tk.IntVar(),
    "login": tk.StringVar(),
    "hasło": tk.StringVar(),
    "regulamin": tk.BooleanVar()
}
plec = tk.StringVar()

# Zakładka 1
frame1 = ttkb.Frame(notebook)
frame1.pack(fill='both', expand=True)
container1 = ttkb.Frame(frame1)
container1.pack(expand=True)

ttkb.Label(container1, text="Imię:").grid(row=0, column=0, sticky="e", padx=5, pady=10)
ttkb.Entry(container1, textvariable=entries["imię"]).grid(row=0, column=1, pady=10)

ttkb.Label(container1, text="Nazwisko:").grid(row=1, column=0, sticky="e", padx=5,
pady=10)
ttkb.Entry(container1, textvariable=entries["nazwisko"]).grid(row=1, column=1, pady=10)

ttkb.Label(container1, text="Płeć:").grid(row=2, column=0, sticky="e", padx=5, pady=10)
ttkb.Radiobutton(container1, text="M", variable=plec, value="M").grid(row=2, column=1,
sticky="w", pady=10)
ttkb.Radiobutton(container1, text="K", variable=plec, value="K").grid(row=2, column=1,
padx=50, sticky="w", pady=10)

ttkb.Label(container1, text="Data urodzenia:").grid(row=3, column=0, sticky="e", padx=5,
pady=10)
ttkb.Entry(container1, textvariable=entries["data"]).grid(row=3, column=1, pady=10)

ttkb.Label(container1, text="Kraj:").grid(row=4, column=0, sticky="e", padx=5, pady=10)
ttkb.Combobox(container1, values=["Polska", "Niemcy", "Francja"],
textvariable=entries["kraj"]).grid(row=4, column=1, pady=10)

# Zakładka 2
frame2 = ttkb.Frame(notebook)
frame2.pack(fill='both', expand=True)
container2 = ttkb.Frame(frame2)
container2.pack(expand=True)

ttkb.Checkbutton(container2, text="Subskrybuj newsletter",
variable=entries["newsletter"]).pack(pady=10)
ttkb.Label(container2, text="Ulubiony kolor:").pack(pady=10)
ttkb.Combobox(container2, values=["Czerwony", "Zielony", "Niebieski"],
textvariable=entries["kolor"]).pack(pady=10)
ttkb.Label(container2, text="Głośność powiadomień:").pack(pady=10)
ttkb.Scale(container2, from_=0, to=100, orient='horizontal',
variable=entries["głośność"]).pack(pady=10)

# Zakładka 3
frame3 = ttkb.Frame(notebook)
frame3.pack(fill='both', expand=True)
container3 = ttkb.Frame(frame3)
container3.pack(expand=True)

ttkb.Label(container3, text="Login:").grid(row=0, column=0, sticky="e", padx=5, pady=10)
ttkb.Entry(container3, textvariable=entries["login"]).grid(row=0, column=1, pady=10)

ttkb.Label(container3, text="Hasło:").grid(row=1, column=0, sticky="e", padx=5, pady=10)
ttkb.Entry(container3, textvariable=entries["hasło"], show="*").grid(row=1, column=1,
pady=10)

ttkb.Checkbutton(container3, text="Akceptuję regulamin",
variable=entries["regulamin"]).grid(row=2, column=1, pady=10)

btn_frame = ttkb.Frame(container3)
btn_frame.grid(row=3, column=1, pady=10)
ttkb.Button(btn_frame, text="Zapisz", command=zapisz, bootstyle="success").grid(row=0,
column=0, padx=5)
ttkb.Button(btn_frame, text="Szukaj", command=szukaj, bootstyle="info").grid(row=0,
column=1)

notebook.add(frame1, text="Dane osobowe")
notebook.add(frame2, text="Preferencje")
notebook.add(frame3, text="Konto")

root.mainloop()
```



C# - alternatywa dla Python na egzaminie

W większości zadań Python może być zastąpiony językiem C#.
W praktyce oznacza to możliwość korzystania z Visual Studio, Windows Forms oraz NET MAUI jako możliwość emulacji na każdym typie urządzenia.

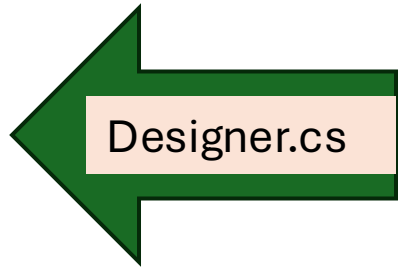
Proste okno z użyciem C# i Windows Forms

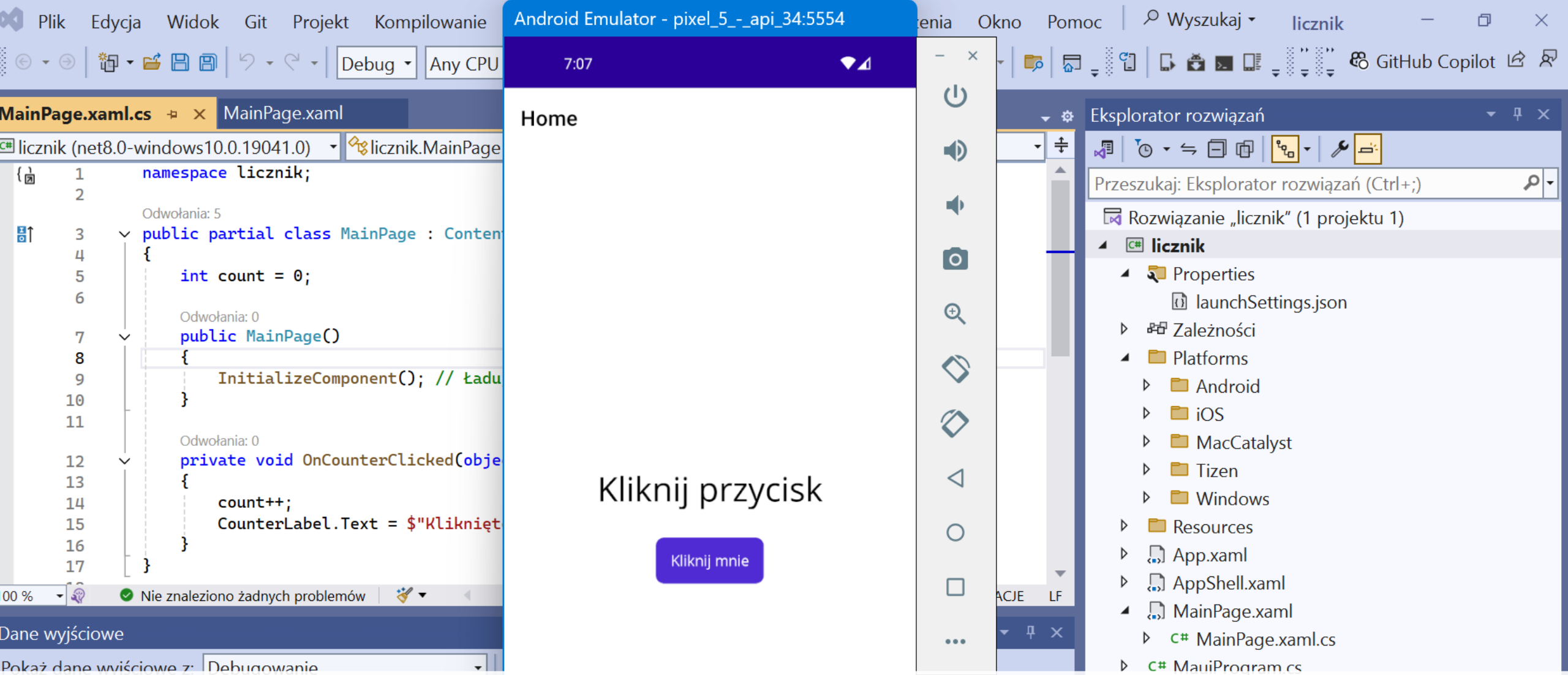
```
using System;
using System.Windows.Forms;


static class Program
{
  [STAThread]
  static void Main()
  {
    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(false);
    Application.Run(new MainForm());
  }
}
```


```
using System;
using System.Windows.Forms;

public partial class MainForm : Form
{
  public MainForm()
  {
    InitializeComponent();  // Inicjalizacja kontrolek (z Designer.cs)
  }

  private void btnClickMe_Click(object sender, EventArgs e)
  {
    lblInfo.Text = "Przycisk został kliknięty!";
  }
}
```

Proste okno z użyciem C# i Windows Forms

```
partial class MainForm
{
    private System.ComponentModel.IContainer components = null;
    private Button btnClickMe;
    private Label lblInfo;

    // Metoda czyszcząca zasoby
    protected override void Dispose(bool disposing)
    {
        if (disposing && (components != null))
        {
            components.Dispose();
        }
        base.Dispose(disposing);
    }

    private void InitializeComponent()
    {
        this.btnClickMe = new System.Windows.Forms.Button();
        this.lblInfo = new System.Windows.Forms.Label();
        this.SuspendLayout();
        //
        // btnClickMe
        //
        this.btnClickMe.Location = new System.Drawing.Point(50, 50);
        this.btnClickMe.Name = "btnClickMe";
        this.btnClickMe.Size = new System.Drawing.Size(100, 30);
        this.btnClickMe.TabIndex = 0;
        this.btnClickMe.Text = "Kliknij mnie";
        this.btnClickMe.UseVisualStyleBackColor = true;
        this.btnClickMe.Click += new
System.EventHandler(this.btnClickMe_Click);
        //
        // lblInfo
        //
        this.lblInfo.AutoSize = true;
        this.lblInfo.Location = new System.Drawing.Point(50, 100);
        this.lblInfo.Name = "lblInfo";
        this.lblInfo.Size = new System.Drawing.Size(0, 15);
        this.lblInfo.TabIndex = 1;
        //
        // MainForm
        //
        this.AutoScaleDimensions = new System.Drawing.SizeF(7F, 15F);
        this.AutoScaleMode =
System.Windows.Forms.AutoScaleMode.Font;
        this.ClientSize = new System.Drawing.Size(300, 200);
        this.Controls.Add(this.lblInfo);
        this.Controls.Add(this.btnClickMe);
        this.Name = "MainForm";
        this.Text = "Przykład WinForms";
        this.ResumeLayout(false);
        this.PerformLayout();
    }
}
```





.NET MAUI – możliwość wykorzystania dowolnego emulatora

Koniec?

Prezentacja zawiera najważniejsze elementy, z których można wykonać większość zadań na egzaminie. Zadania egzaminacyjne z aplikacji desktopowych zawierają zwykle 3 elementy: wpisywanie danych, zapis i wyświetlenie w postaci tabeli lub listy oraz możliwość ich wyszukiwania.

Częstym elementem są układy widgetów oraz różnego rodzaju tła. Większość tych zadań ma formę jednego pliku. Jednak bardziej właściwym będzie wykonanie oddzielnego pliku z oknem programu oraz drugiego ze sposobem działania.