# Projekt zaliczeniowy – Języki skryptowe (Python)

Tytuł projektu: System opieki nad zwierzętami w schronisku

Autorzy: Jakub Kosatka, Oleksandr Stankevych

Grupa: Informatyka, 2ID12B  
Data oddania: 26.06.2025

## 1. Cel projektu

Celem projektu było stworzenie aplikacji w języku Python do systemu opieki nad zwierzętami w schronisku. Aplikacja umożliwia dodawanie, edytowanie, usuwanie, adopcje na obiektach zwierzęta. Projekt łączy programowanie obiektowe (OOP) i funkcyjne, z naciskiem na obsługę błędów, testowanie jednostkowe oraz modularną strukturę kodu.

## 2. Zakres funkcjonalny

Stworzenie systemu do opieki nad zwierzętami w schronisku, który pozwala na:

* Dodawanie nowych zwierząt do listy z uwzględnieniem imieniu, wieku, gatunku zwierzęta oraz statusu zaszczepienia.
* Edytowanie istniejących zwierząt(imię, wiek, gatunku, statusu zaszczepienia i adoptowania).
* Usunięcie zwierząt z listy
* Wyszukanie zwierząt z listy
* Adaptowanie zwierzęta przez osobę (imię, nazwisko, pesel, numer telefonu)
* Nakarmienie zwierzęta
* Wizualizację rozkładu zwierząt (gatunek, szczepienia) z użyciem matplotlib.
* Zapis i odczyt danych w formacie JSON.

### Funkcje aplikacji

* **Dodawanie zwierzęta**: Użytkownik może dodać zwierze z imieniem, wiekiem, gatunkiem (kot,pies,królik itd), i statusem zaszczepienia
* **Edytowanie zwierzęta**: Możliwość zmiany imieniu, wieku, gatunkuu, statusu zaszczepienia i adoptowania.
* **Usunięcie zwierzęta**: Usunięcie zwierzęcia z listy zwierząt (jeżeli nie jest adaptowany) lub z listy adopcji
* **Wyszukiwanie zwierzęta**: Wyszukiwanie zwierząt z listy wykorzystując filtry do wyszukiwania (imię, wiek, gatunek, status zaszczepienia, status adoptowania, data przyjęcia)
* **Adaptowanie zwierzęta**: Adoptowanie zwierzęta przez osobe i dodawanie zwierzęta do listy zwierząt adoptowanych
* **Tworzenie Raportu**: Generowanie wykresów rozkładu zwierzęta bazujących na gatunku lub informacji o szczepienia.
* **Zapis danych**: Zapisywanie danych o zwierzętach do pliku JSON i ich wczytywanie.

### Zakres funkcjonalny

* Bazowa klasa Animal i klasy potomne (Dog, Cat, Bird, Rabbit, Hamster, Turtle) ze wspólną obsługą daty przyjęcia, szczepień, adopcji i ostatniego karmienia. Klasa DataManager do zarządzania danymi, ich ładowanie, zapis. Klasa ShelterApp odpowiadajaca za funkcje programu (interfejsu graficznego, operacje na obiektach klasy Animal i klasy DataManager)
* Interfejs graficzny – dwa widoki (Lista zwierząt, Adopcje) z paginacją, sortowaniem kolumn w trzech stanach (rosnący,malejący).
* Walidacja danych z assert i try-except (np. poprawny PESEL, telefon, formaty dat, dodatni wiek).
* Zarządzanie danymi (DataManager) - odczyt / zapis do JSON (zwierzęta + adopcje) wraz z pamiętaniem kolejnych ID,import / eksport CSV z walidacją wierszy i szczegółowym raportem błędów,mapowanie nazw gatunków na klasy oraz synchronizacja statusów adopcji.
* Wizualizacja danych – interaktywne słupkowe / kołowe wykresy liczby szczepień lub struktury gatunkowej z możliwością zapisania do PNG.
* Dekorator @log\_action – punkt zaczepienia pod późniejsze logowanie/telemetrię akcji użytkownika.
* Selektywne użycie filter, lambda, oraz rekurencyjnej funkcji zliczającej zwierzęta.
* Testy jednostkowe, funkcjonalne, integracyjne, wydajnościowe i pamięci (unittest/pytest + memory\_profiler).

## 3. Struktura projektu

* **main.py**: Główny moduł uruchamiajacy aplikację i definiuje klasę ShelterApp z całym interfejsem GUI
* **animal\_manager.py**: Zawiera klasy definicje klas Animal, Dog, Cat, Bird, Rabbit, Hamster, Turtle.
* **data\_manager.py**: Moduł z funkcjami do wczytywania i zapisywania danych w formacie JSON/CSV.
* **visualization.py**: Moduł z funkcją plot\_rent\_distribution do generowania wykresów rozkładu czynszu za pomocą matplotlib.
* **decorators.py**: dekorator log\_action dla logowania akcji .
* **zwierzeta.json / adopcje.json**: pliki danych tworzone podczas pracy programu (domyślnie generowane, jeśli brak).
* **Adopcje\_przyk.csv/Zwierzeta\_przyk.csv**: Pliki używany w testach (przykładowe dane)
* **test.py:** Zbiór testów jednostkowych, funkcjonalnych, integracyjnych, wydajnościowych i pamięci.

### Krótkie omówienie każdej klasy/modułu

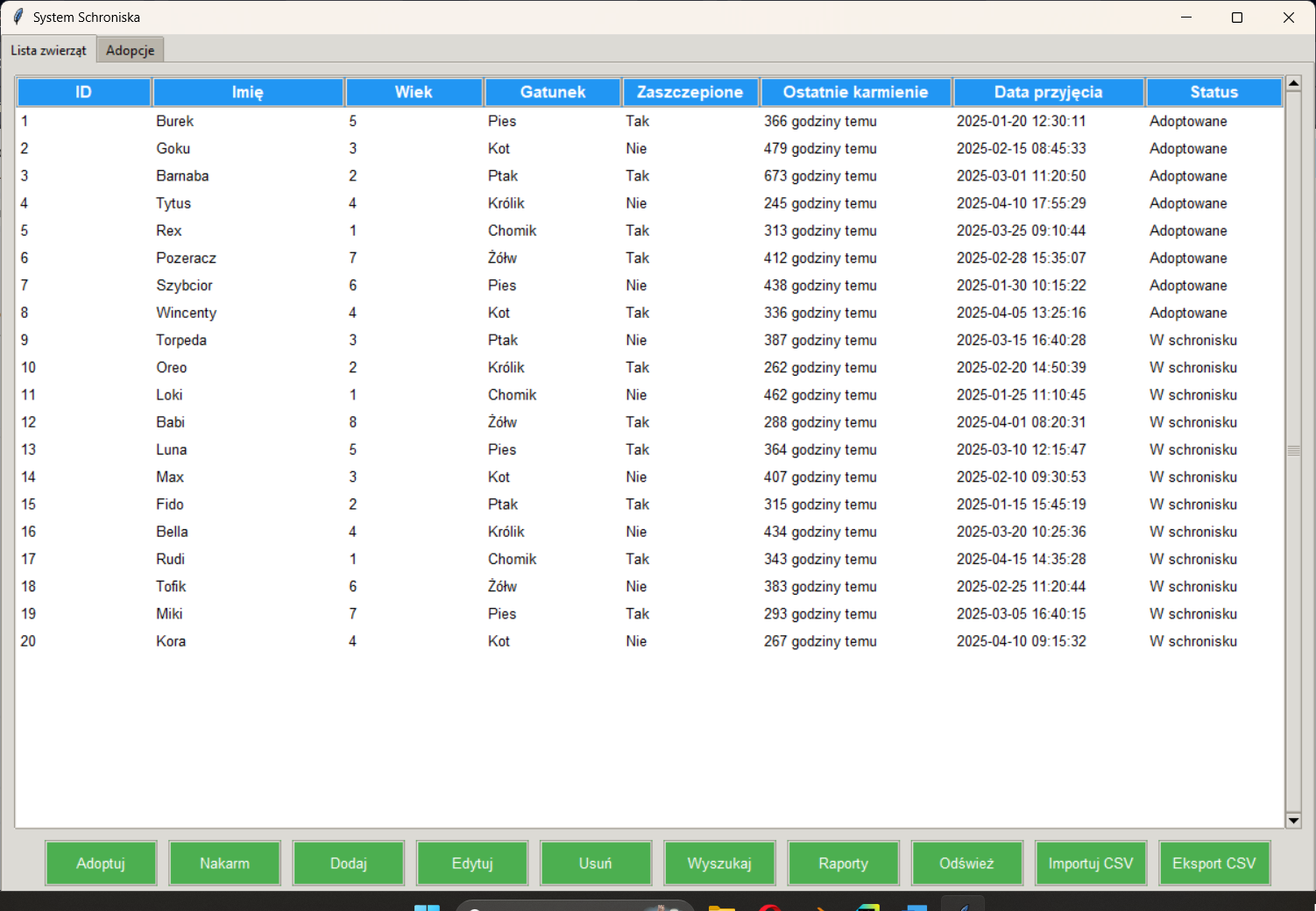
* **Animal**: wspólna baza atrybutów i metod (ID, imię, wiek, daty, flagi). Metoda get\_feeding\_status() wylicza ile godzin minęło od ostatniego karmienia.
* **Dog/Cat/Bird/Rabbit/Hamster/Turtle**: puste klasy dziedziczące po Animal (pozwalają rozróżniać gatunki zwierzęt).
* **DataManager**: Klasa dziedzicząca po Pokoj, dodająca listę udogodnień (np. WiFi, TV).
* **App**: Główna klasa aplikacji, zarządzająca listą pokoi, interakcją z użytkownikiem i operacjami zapisu/odczytu danych. Implementuje menu, dodawanie, edytowanie, filtrowanie i wizualizację.
* **utils.py**: Funkcje load\_data i save\_data do obsługi plików JSON.
* **visualization.py**: Funkcja plot\_rent\_distribution generująca histogram czynszu.
* **test.py**: Testy weryfikujące poprawność klas, metod, obsługi błędów i wydajności.

## 4. Technologie i biblioteki

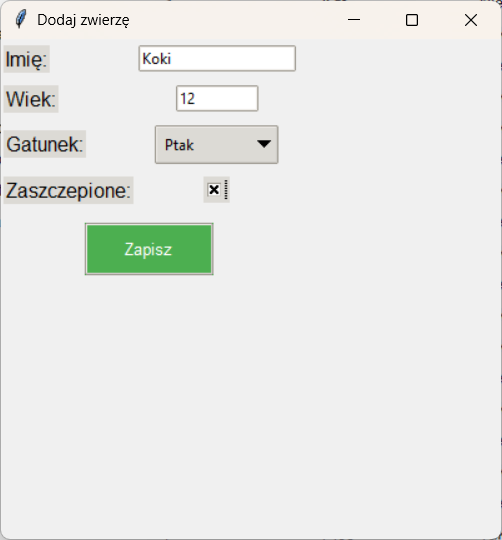
* **Python 3.10+**
* **json/csv**: trwały zapis danych (JSON) + import/eksport (CSV).
* **Tkinter / ttk / tkcalendar**: interfejs graficzny, komponenty kalendarza do wyboru dat.
* **functools.wraps**: implementacja dekoratora log\_action.
* **datetime**: operacje na datach, walidacje formatów, obliczanie różnic czasu.
* **matplotlib**: generowanie wykresów w raportach.
* **unittest / pytest**: frameworki do testów.
* **memory\_profiler**: Do analizy zużycia pamięci w teście test\_memory\_save.

## 5. Sposób działania programu

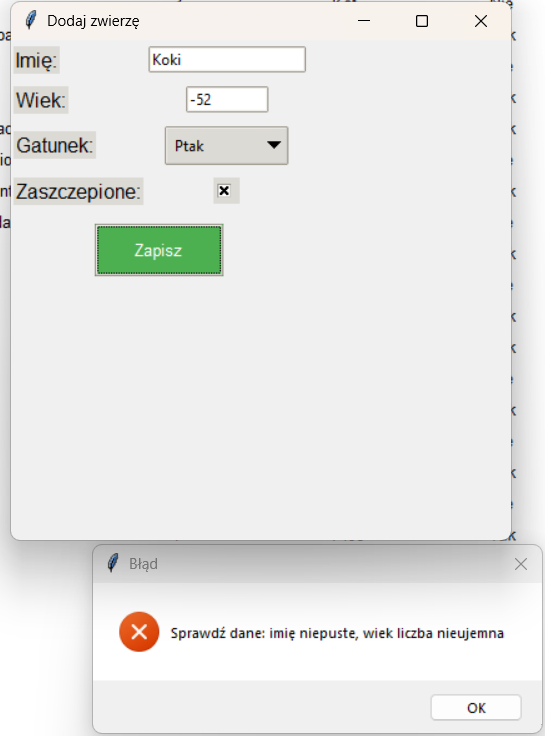
Aby skorzystać z programu, należy uruchomić plik Schronisko.exe



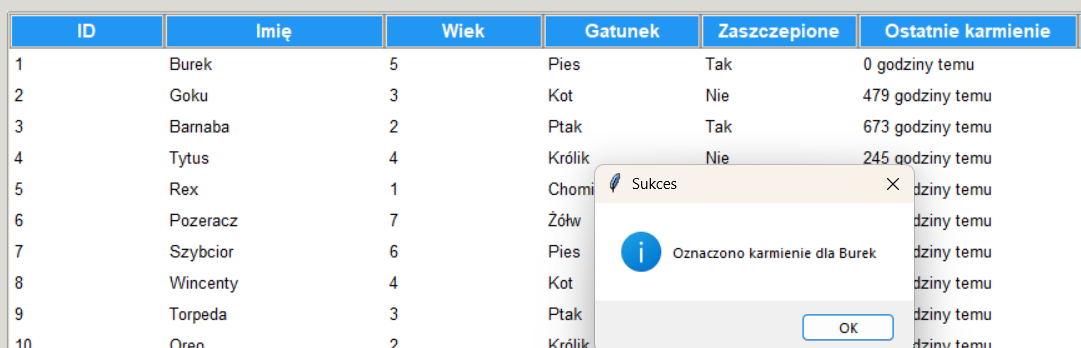
rys 1. Wygląd programu: Okno Lista zwierząt



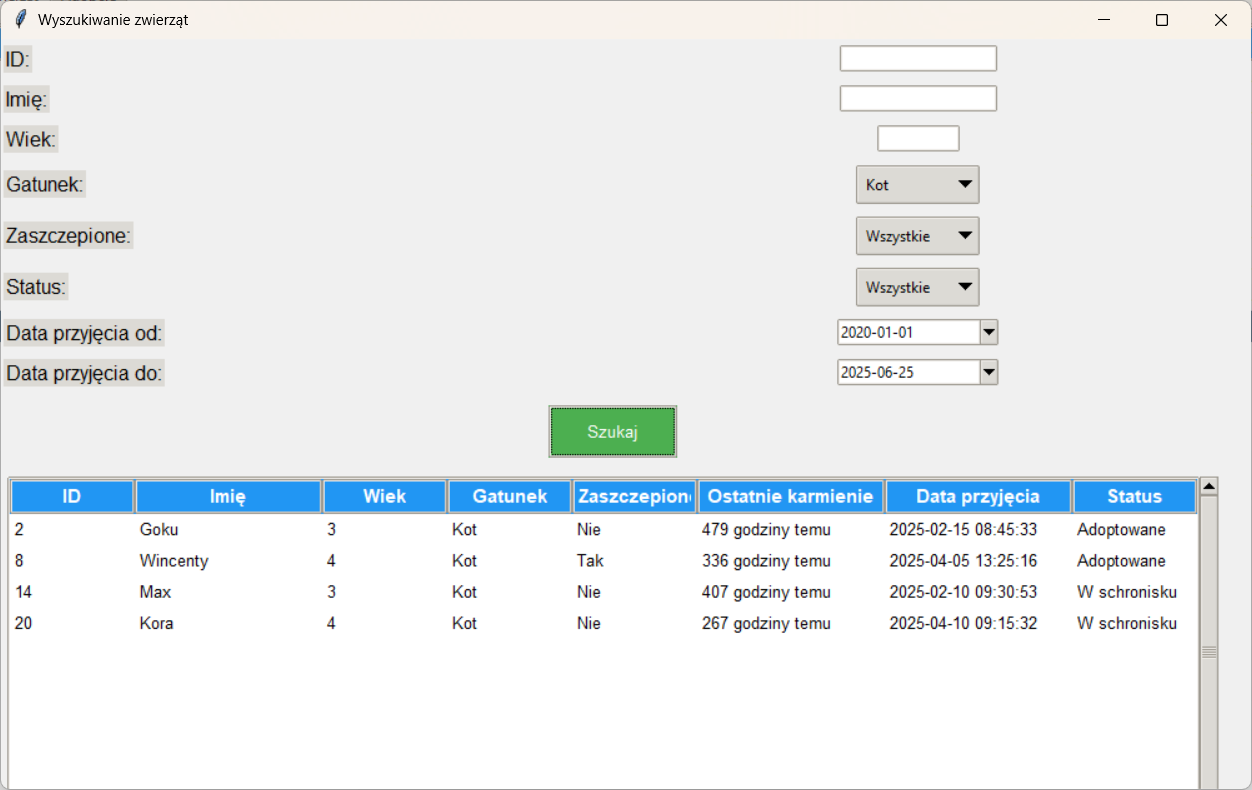
rys 2. Okno z dodawaniem zwierzęta do listy



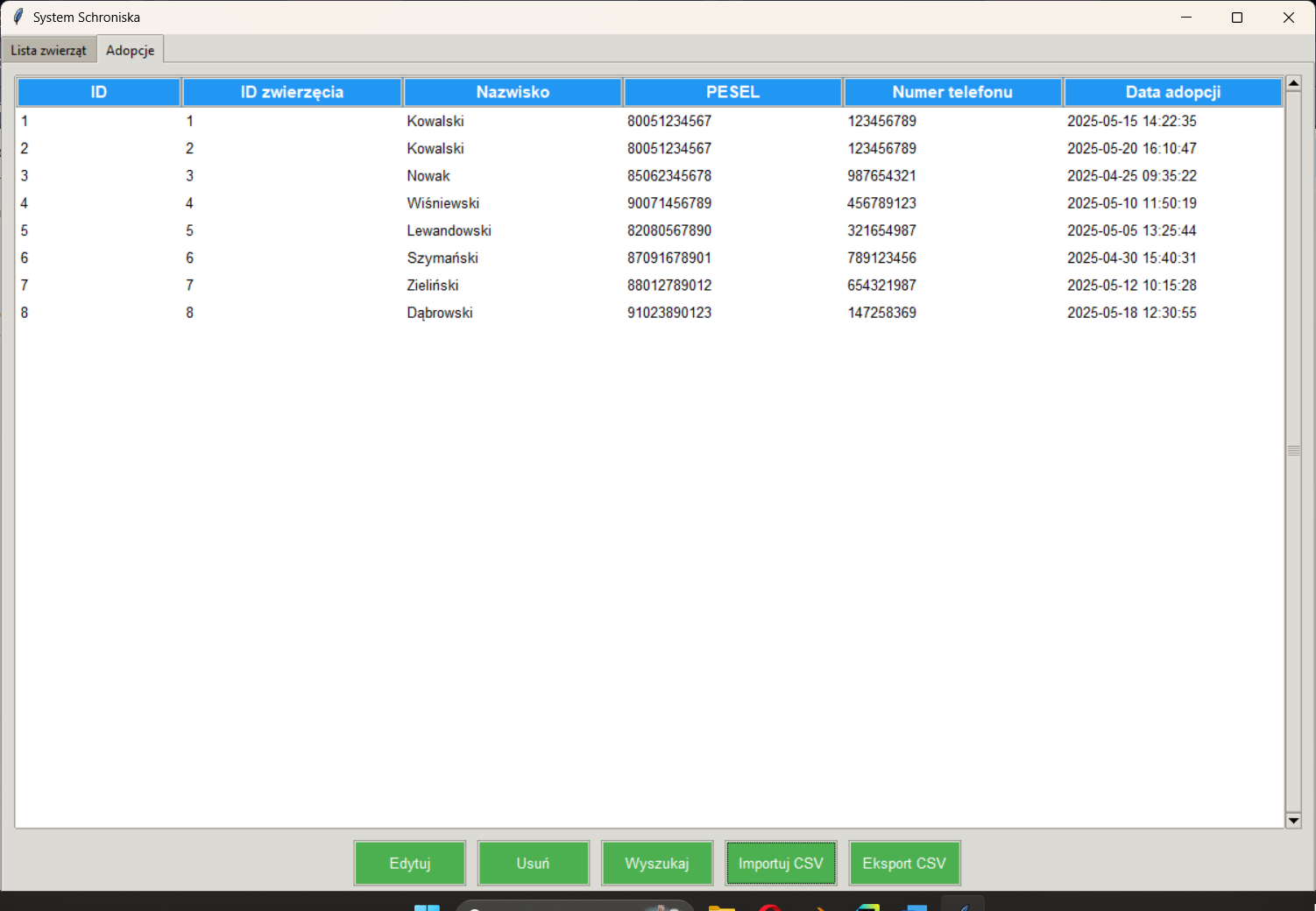
rys 3. Komunikat o błędzie (Wiek jest ujemny)



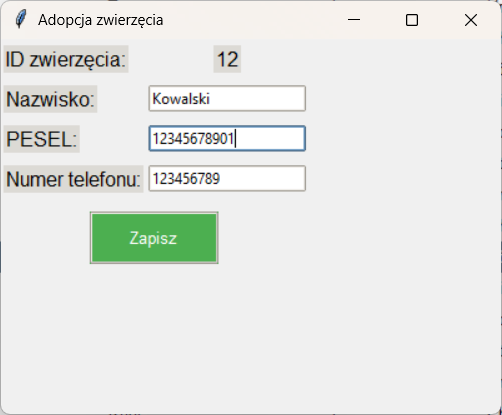
rys 4. Komunikat o karmeniu



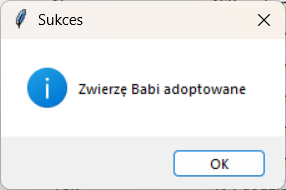
rys 5. Okno z wyszukaniem



rys 6. Lista adopcji



rys 7. Okno adopcji



rys 8. Komunikat o adoptowaniu



rys 9. Stworzony raport na podstawie gatunków

## 

## 6. Przykłady kodu

### 

### Fragment funkcji funkcyjnej

def generate\_report():

animals\_list = list(self.animals.items())

vaccinated = len(list(filter(lambda x: x[1].is\_vaccinated and not x[1].is\_adopted, animals\_list)))

not\_vaccinated = len(list(filter(lambda x: not x[1].is\_vaccinated and not x[1].is\_adopted, animals\_list)))

W jednym wyrażeniu filter + lambda odrzuca rekordy, które nie spełniają podanych warunków. Dwa wywołania zwracają metryki do raportu: ile zwierząt jest zaszczepionych, a ile nie, przy czym oba zbiory pomijają już adoptowane zwierzęta. Taki funkcyjny zapis zastępuje pętle for i skraca kod.

### 

### Fragment klasy

class Animal:

# Inicjalizacja zwierzęcia z podstawowymi atrybutami

def \_\_init\_\_(self, id, name, age):

self.id = id

self.name = name

self.age = age

self.is\_adopted = False

self.is\_vaccinated = False

self.last\_fed = None

self.admission\_date = None

# Zwraca status ostatniego karmienia

def get\_feeding\_status(self):

if not self.last\_fed:

return "Brak danych"

try:

last\_fed\_time = datetime.strptime(self.last\_fed, "%Y-%m-%d %H:%M:%S")

delta = datetime.now() - last\_fed\_time

hours = int(delta.total\_seconds() // 3600)

return f"{hours} godziny temu"

except ValueError:

return "Nieprawidłowy format daty"

Klasa Animal gromadzi wspólne pola wszystkich gatunków, a get\_feeding\_status() zwraca informację, ile godzin minęło od ostatniego karmienia. try-except chroni przed błędnym formatem daty, więc metoda nigdy nie przerwie działania aplikacji, a klasy potomne dziedziczą gotową, odporną na błędy logikę.

### 

### 

### Obsługa wyjątków

def import\_animals\_csv(self, file\_path, replace=True):

errors = []

new\_animals = {} if replace else self.animals.copy()

new\_next\_id = 1 if replace else self.next\_id

try:

with open(file\_path, 'r', encoding='utf-8') as f:

reader = csv.reader(f, delimiter=';')

headers = next(reader)

if headers != ["ID", "Imię", "Wiek", "Gatunek", "Zaszczepione", "Ostatnie karmienie", "Data przyjęcia", "Status"]:

errors.append("Nieprawidłowe nagłówki pliku CSV")

return errors

row\_index = 0

while row\_index < len(list(reader)):

f.seek(0)

reader = csv.reader(f, delimiter=';')

next(reader)

for row\_index, row in enumerate(reader):

row\_errors = []

try:

animal\_id = row[0].strip()

name = row[1].strip()

age = int(row[2].strip()) if row[2].strip() else 0

species = row[3].strip()

is\_vaccinated = row[4].strip().lower() == "tak"

last\_fed = row[5].strip() if row[5].strip() else None

admission\_date = row[6].strip() if row[6].strip() else None

is\_adopted = row[7].strip().lower() == "adoptowane"

if not name:

row\_errors.append("Imię nie może być puste")

if age < 0:

row\_errors.append("Wiek musi być nieujemny")

if species not in self.species\_map:

row\_errors.append(f"Nieprawidłowy gatunek: {species}")

if last\_fed:

try:

datetime.strptime(last\_fed, "%Y-%m-%d %H:%M:%S")

except ValueError:

row\_errors.append(f"Nieprawidłowy format daty ostatniego karmienia: {last\_fed}")

if admission\_date:

try:

datetime.strptime(admission\_date, "%Y-%m-%d %H:%M:%S")

except ValueError:

row\_errors.append(f"Nieprawidłowy format daty przyjęcia: {admission\_date}")

if animal\_id in new\_animals:

row\_errors.append(f"Powielone ID zwierzęcia: {animal\_id}")

if not row\_errors:

animal = self.species\_map[species](animal\_id, name, age)

animal.is\_vaccinated = is\_vaccinated

animal.last\_fed = last\_fed

animal.admission\_date = admission\_date

animal.is\_adopted = is\_adopted

new\_animals[animal\_id] = animal

new\_next\_id = max(new\_next\_id, int(animal\_id) + 1)

else:

errors.append(f"Wiersz {row\_index + 2}: {', '.join(row\_errors)}")

except Exception as e:

errors.append(f"Wiersz {row\_index + 2}: Błąd: {str(e)}")

self.animals = new\_animals

self.next\_id = new\_next\_id

self.save\_animals(self.animals, self.next\_id, self.next\_adoption\_id)

except Exception as e:

errors.append(f"Błąd importu: {str(e)}")

return errors

Metoda importu otwiera plik CSV i sprawdza zgodność nagłówków. Każdy błąd (brak pliku, zły nagłówek, zły typ danych) łapią bloki except, a szczegóły trafiają do listy errors. Dzięki temu użytkownik dostaje pełny raport, a GUI nie wyświetla surowego tracebacka.

## 

## 7. Testowanie

### Opis sposobu testowania

Testy zostały zaimplementowane w module test.py z użyciem biblioteki unittest oraz dekoratora @profile z memory\_profiler.. Obejmują:

* **Testy jednostkowe**: test\_get\_feeding\_status sprawdza poprawność metody get\_feeding\_status w klasie Animal, natomiast test\_save\_load\_animals weryfikuje, że DataManager prawidłowo zapisuje i wczytuje strukturę JSON.
* **Testy funkcjonalne**: test\_functional\_add\_animal symuluje dodanie nowego zwierzęcia przez GUI i potwierdza, że rekord pojawia się w widoku Treeview.
* **Testy integracyjne**: test\_integration\_export\_import\_csv wykonuje pełny cykl eksport CSV - reset danych - import CSV, sprawdzając zachowanie identycznych rekordów po odtworzeniu.
* **Testy graniczne**: test\_invalid\_age, test\_invalid\_species, test\_invalid\_date\_format i test\_duplicate\_id badają reakcję systemu na odpowiednio: ujemny wiek, nieobsługiwany gatunek, błędny format daty oraz duplikat identyfikatora.
* **Test wydajności**: test\_performance\_save\_animals mierzy czas wielokrotnego zapisu 1 000 rekordów i wymaga wyniku poniżej 1 s.
* **Test pamięci**: test\_memory\_save\_animals z memory\_profiler analizuje szczytowe zużycie RAM podczas zapisu dużej listy zwierząt, wychwytując ewentualne wycieki.

### Obsługa przypadków granicznych

* **Import pliku CSV z błędnymi danymi**: w teście test\_import\_invalid\_csv tworzony jest sztuczny plik CSV, w którym rekord ma puste imię, ujemny wiek, nieznany gatunek oraz niepoprawne formaty dat.

## 8. Wnioski

Projekt schroniskowy pozwolił w praktyce przećwiczyć pełne spektrum Pythona – od OOP i GUI (Tkinter) przez walidację danych oraz styl funkcyjny po test-driven development – pokazując korzyści z wyraźnego oddzielenia logiki biznesowej od warstwy prezentacji, wczesnej kontroli błędów przy imporcie CSV i pakietu testów wydajnościowych oraz pamięci, a jednocześnie tworząc elastyczną bazę, którą można łatwo rozbudować o REST-owe API, raporty PDF czy moduł płatności, aby przekształcić ją w kompleksową platformę wspierającą wiele schronisk.

GitHub: <https://github.com/KosaK27/ProjektJS>