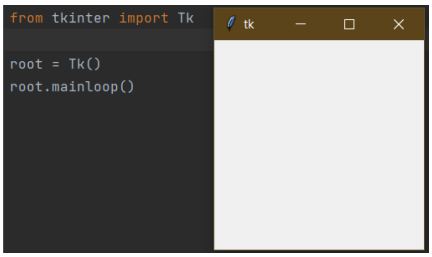
GUI – Tkinter, baza danych,obsługa wyjątków

Tkinter jest wbudowaną biblioteką pythona, zapewniająca zestaw narzędzi do budowania aplikacji z graficznym interfejsem użytkownika.
Dokumentacja tkinter:
https://tkdocs.com/index.html
Aby utworzyć proste okno wystarczą 3 linie kodu:

```
import tkinter as tk
root = tk.Tk()
root.mainloop()
```

Tworzymy główne okno programu „root” ,które będzie rodzicem dla wszystkich innych elementów interfejsu użytkownika. Aby aplikacja uruchomiła się należy wywołać główną pętle zdarzeń.

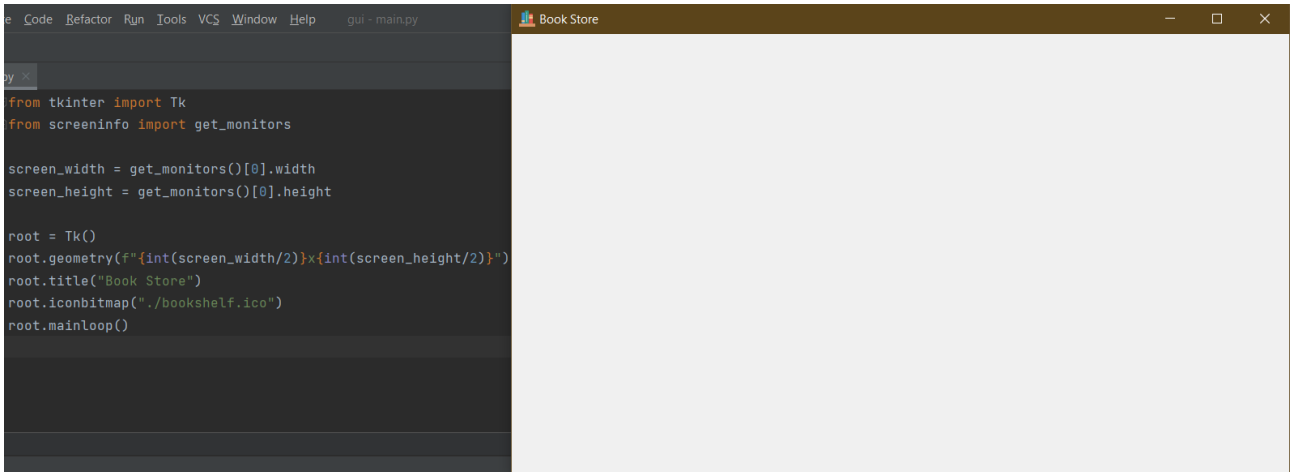


Możemy ustawić kilka dodatkowych właściwości naszego głównego okna – tytuł, wysokość oraz szerokość, a nawet zmienić ikonkę aplikacji:

```
root.title("Book Store")
root.geometry("800x600")
root.iconbitmap("./bookshelf.ico")
```

Jeżeli nie chcemy ręcznie ustawiać wymiarów, możemy wykorzystać rozmiar ekranu, np:

```
from screeninfo import get_monitors

screen_width = get_monitors()[0].width
screen_height = get_monitors()[0].height
root.geometry(f"{int(screen_width/2)}x{int(screen_height/2)}")
```



Używając tkinter możemy wykorzystać wiele wbudowanych kontrolek-widżetów.

Przykładowe kontrolki:
Przycisk (Button): Pozwala na interakcję użytkownika poprzez kliknięcie.
Etykieta (Label): Służy do wyświetlania tekstu lub obrazków.
Pole tekstowe (Entry): Umożliwia użytkownikowi wprowadzanie tekstu.
Pole wyboru (Checkbutton): Pozwala na zaznaczanie lub odznaczanie opcji.
Przycisk opcji (Radiobutton): Pozwala na wybór jednej z kilku opcji.
Lista rozwijana (Combobox): Wyświetla rozwijaną listę opcji do wyboru.
Suwak (Scale) umożliwia użytkownikowi wybór wartości z zakresu, przesuwając suwak wzdłuż osi.
Pola tekstowe wielolinijkowe (Text): Służą do wprowadzania i wyświetlania tekstu na wielu liniach.

Napiszmy pierwszą funkcjonalność – zmiana tekstu etykiety za pomocą pola tekstowego i przycisku:

Aby utworzyć nowy widget tworzymy nową instancję podając rodzica nowego widżetu i opcjonalnie tekst, który będzie wyświetlany:

```
label = tk.Label(root, text="Podaj imię i nazwisko:")
label.pack()
```

Aby dodać widżet do okna w Tkinterze, możemy użyć metody pack(), grid() lub place() na obiekcie widżetu.
pack() - widżety są domyślnie umieszczane jeden pod drugim lub jeden obok drugiego.
grid() - widżety są umieszczane w odpowiednich wierszach i kolumnach.
place() - widżety rozmieszczane są w oknie, korzystając z określonych współrzędnych x i y.

Pole tekstowe:

```
entry = tk.Entry(root)
entry.pack()
```

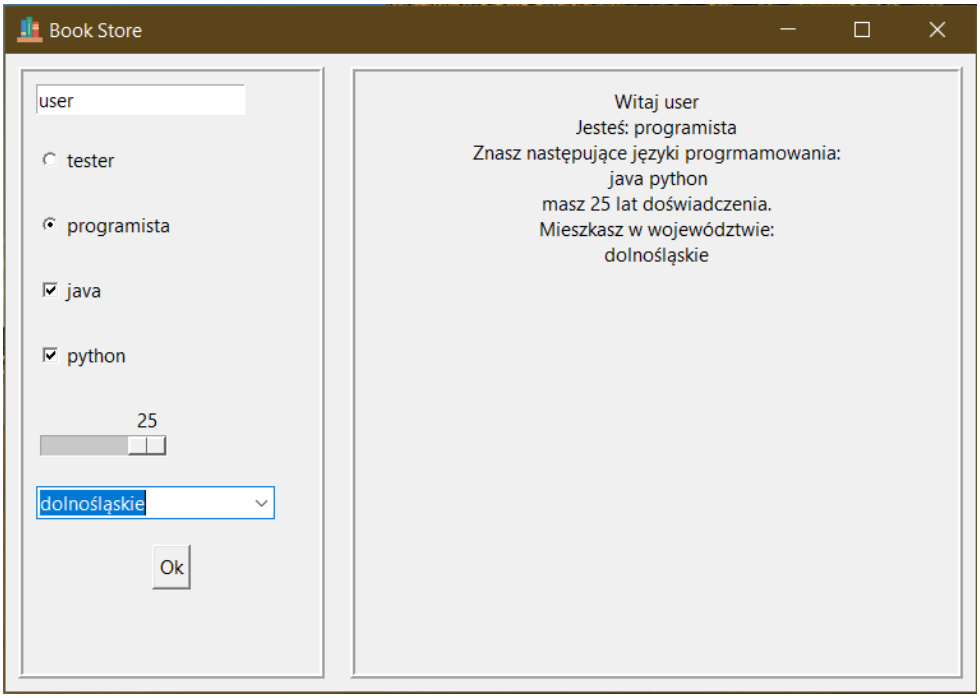
Przycisk:

```
button = tk.Button(root, text="Ok", command=change_text)
```

Parametr command jest używany do przypisania funkcji, która zostanie wykonana po kliknięciu przycisku.

entry.get() pobiera tekst z pola tekstowego, a label.config(text) ustawia tekst etykiety.
Funkcja może wyglądać następująco:

```
def change_text():
    name= entry.get()  # Pobranie tekstu z pola tekstowego
    label.config(text="Witaj "+name)  # Zmiana tekstu etykiety na nowy tekst
```



Pobierzemy więcej informacji od użytkownika, interaktywne widżety umieścimy po lewej stronie. Tekst w postaci etykiety label wyświetlimy po prawej.

Utworzymy dwie ramki (Frame) z rodzicem będącym naszym oknem głównym(root) po lewej i prawej stronie. Zamiast rozmiaru okna ustawimy rozmiar ramek (width, height) wraz z rozmiarem i typem obramowania(borderwidth,relief), ustawimy po której stronie okna będą się znajdować (side), ustawimy odstęp od okna (padx,pady) oraz nie pozwolimy na zmianę ich rozmiaru .pack_propagate(0)

```
left_frame = tk.Frame(root,borderwidth=4, relief="ridge",width=int(screen_width / 8),height=int(screen_width / 4))
left_frame.pack(side="left",padx=10, pady=10)
left_frame.pack_propagate(0)

right_frame = tk.Frame(root,width=int(screen_width / 4), height=left_frame["height"],borderwidth=4, relief="ridge")
right_frame.pack(side="right",padx=10, pady=10)
right_frame.pack_propagate(0)
```

Aby uniemożliwić zmianę rozmiaru okna użyjemy:

```
root.resizable(width=False, height=False)
```

Widżety nie będą już w głównym oknie a w lewej ramce (anchor- określa gdzie widżet będzie przypięty, przyjmuje wartości reprezentujące kierunki na kompasie oraz środek):

```
entry = tk.Entry(left_frame)
entry.pack(anchor="w",padx=10, pady=10)

button = tk.Button(left_frame, text="Ok", command=change_text)
button.pack(anchor="center",padx=10, pady=10)
```

Etykieta po prawej stronie:

```
label = tk.Label(right_frame, text="podaj dane")
label.pack(padx=10, pady=10)
```

Radiobutton będzie służył do jednokrotnego wyboru (programista/tester).

Do przechowywania i zarządzania wartościami tekstowymi w widżetach Tkinter służy klasa StringVar.
Dodajmy wartość którą będziemy zmieniali dokonując wyboru (w tym wypadku dopiero po naciśnięciu przycisku Ok)

```
radio_var = tk.StringVar(value=" ")
```

Domyślnie ustawimy wartość na „ ” - początkowo nie wybraliśmy nic.
Radiobuttony:

```
radio_button1 = tk.Radiobutton(left_frame, text="tester", variable=radio_var, value="testerem")
radio_button1.pack(anchor="w",padx=10, pady=10)

radio_button2 = tk.Radiobutton(left_frame, text="programista", variable=radio_var, value="programista")
radio_button2.pack(anchor="w",padx=10, pady=10)
```

Gdy wciśniemy przycisk ok wyświetlimy informację o dokonanym wyborze w funkcji change_text:

```
    new_text = "Witaj "+entry.get()
    who = "\nJesteś: "+radio_var.get()

    label.config(text=new_text + who)
```

Do wyboru znajomości języka użyjemy checkbutton’ów, do ich przechowywania, oraz do przechowywania informacji o tym czy zostały zaznaczone użyjemy list:

```
checkbox_vars = []
checkboxes = []
```

Tym razem będziemy używali wartości liczbowej tk.IntVar() : 1 - checkbutton został zaznaczony.

```
checkbox_vars.append(tk.IntVar())
checkbox = tk.Checkbutton(left_frame, text="java", variable=checkbox_vars[0])
checkbox.pack(anchor="w",padx=10, pady=10)
checkboxes.append(checkbox)
checkbox_vars.append(tk.IntVar())
checkbox2 = tk.Checkbutton(left_frame, text="python", variable=checkbox_vars[1])
checkbox2.pack(anchor="w",padx=10, pady=10)
checkboxes.append(checkbox2)
```

W funkcji change_text sprawdzamy czy checkbutton został zaznaczony (lista checkbox_vars) jeżeli tak to pobieramy z niego tekst (lista checkboxes):

```

languages = "\nZnasz następujące języki progrmamowania:\n"
for i in range(len(checkbox_vars)):
    if checkbox_vars[i].get() == 1:
        languages = languages+" "+checkboxes[i].cget("text")
```

Tekst z widżetu Checkbutton pobieramy za pomocą .cget("text").

Kolejnym elementem jakiego użyjemy jest slider:

```
slider = tk.Scale(left_frame, from_=0, to=25, orient=tk.HORIZONTAL)
slider.pack(anchor="w",padx=10, pady=10)
```

Określamy zakres (from_, to), oraz położenie suwaka(orient).
W funkcji change_text pobieramy wartość slidera:

```
experience = "\nmasz "+ str(slider.get()) + " lat doświadczenia."
```

Ostatnim przykładowym widżetem jest lista rozwijana Combobox. Pochodzi z pakietu ttk:

```
from tkinter import ttk
```

Opcje do wyboru (województwa) umieszczamy w liście:

```
values = [
    "dolnośląskie",
    "kujawsko-pomorskie",
    "lubelskie",
    "lubuskie",
    "łódzkie",
    "małopolskie",
    "mazowieckie",
    "opolskie",
    "podkarpackie",
    "podlaskie",
    "pomorskie",
    "śląskie",
    "świętokrzyskie",
    "warmińsko-mazurskie",
    "wielkopolskie",
    "zachodniopomorskie"
]
```

Następnie tworzymy Combobox z wartościami z listy:

```
combobox = ttk.Combobox(left_frame, values=values)
combobox.state(["readonly"])
combobox.pack(anchor="w",padx=10, pady=10)
```

combobox.state(["readonly"]) uniemożliwia wpisywania innych wartości przez użytkownika.
Pobieranie wybranej wartości i ostateczna zmiana tekstu :

```

home = "\nMieszkasz w województwie:\n"+combobox.get()
label.config(text=new_text + who + languages + experience + home)
```

W dalszej części napiszemy aplikację umożliwiającą przeglądanie,edytowanie,dodawanie i usuwanie książek z bazy danych.

Potrzebujemy dostępu do bazy danych. Dobrym darmowym wyborem jest mysql.
Darmową bazę można założyć na stronie:
https://www.db4free.net/

Po założeniu konta i zalogowaniu się do phpmyadmin możemy utworzyć tabelkę Books i uzupełnić ją danymi. W przykładzie posłużymy się bardzo prostą tabelką, nie tworzymy relacji.

```
use pjatkdbpython;
drop table if exists `books`;
CREATE TABLE books (
  id INT PRIMARY KEY AUTO_INCREMENT,
  title VARCHAR(255) NOT NULL,
  author VARCHAR(255) NOT NULL,
  price DECIMAL(10, 2) NOT NULL,
  category VARCHAR(255) NOT NULL
);
```

```
INSERT INTO books (title, author, price, category)
VALUES
  ('Green Town', 'Ray Bradbury', 30.58, 'Science-Fiction'),
  ('Baśniowa opowieść', 'Stephen King', 30.54, 'Science-Fiction'),
  ('To', 'Stephen King', 42.99, 'Horror');
```

Dane możemy pobrać za pomocą konektora i połączenia z bazą danych(`mysql-connector-python`) oraz kursora. Kursor jest obiektem, który umożliwia interakcję z bazą danych MySQL poprzez wykonywanie zapytań i pobieranie wyników z bazy.

```
def fetch_data():
    mydb = mysql.connector.connect(
        host="db4free.net",
        user="username",
        password="password",
        database="databasename"
    )
    mycursor = mydb.cursor()
    mycursor.execute("SELECT * FROM books")

    result = mycursor.fetchall() #pobiera wszystkie wyniki
    mycursor.close()  # Zamknięcie kursora
    mydb.close()  # Zamknięcie połączenia
    return result
```

Po wywołaniu tej funkcji wynikiem będą:
[(1, 'Green Town', 'Ray Bradbury', Decimal('30.58'), 'Science-Fiction'), (2, 'Baśniowa opowieść', 'Stephen King', Decimal('30.54'), 'Science-Fiction'), (3, 'To', 'Stephen King', Decimal('42.99'), 'Horror')]

Książki możemy umieścić w widgecie Treeview.
Treeview może służyć do wyświetlania danych w formie tabeli tabeli hierarchicznej z zagnieżdżonymi elementami. Aby dodać do niej elementy w prosty sposób utworzymy kolumny i ich tekst zgodnie z danymi z naszej tabeli books:

```
treeview = ttk.Treeview(root)
treeview["columns"] = ("id", "title", "author", "price", "category")
treeview.column("#0", width=0)
treeview.heading("id", text="ID")
treeview.heading("title", text="Tytuł")
treeview.heading("author", text="Autor")
treeview.heading("price", text="Cena")
treeview.heading("category", text="Kategoria")

treeview.pack()
```

treeview.heading() – ustawia nagłówki kolumn.
Napiszmy funkcję, która doda elementy z tabeli books do treeview:

```
def load_data():
    data = fetch_data()
    # Usunięcie poprzednich danych
    treeview.delete(*treeview.get_children())
    # Dodanie nowych danych
    for row in data:
        treeview.insert("", "end", values=(row[0], row[1], row[2], row[3], row[4]))
```

* - rozpakowuje elementy z listy jako kolejne argumenty przekazywane do funkcji delete.

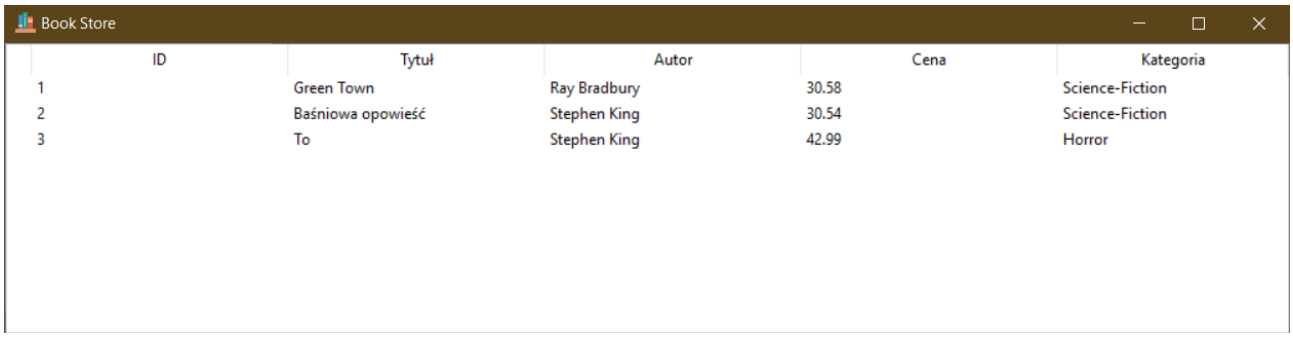
Za pomocą treeview.insert w pętli dodajemy wartości kolumn dla poszególnych wierszy z wyniku naszego zapytania:

```
treeview.insert("", "end", values=(row[0], row[1], row[2], row[3], row[4]))
```

Wartości "", "end" to kolejno parent oraz index. Pierwszy jest pustym stringiem – czyli element nie ma rodzica, w którym mógłby być zagnieżdżony, "end" - to dodawanie elementów na koniec. „0” - dodawałoby elementy na początek.

Utworzenie okna i dodanie treeview wygląda następująco:

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

root = tk.Tk()
root.title("Book Store")
root.iconbitmap("./bookshelf.ico")

# Pobranie danych z tabeli
def fetch_data():
    mydb = mysql.connector.connect(
        host="db4free.net",
        user="username",
        password="password",
        database="databasename"
    )
    mycursor = mydb.cursor()
    mycursor.execute("SELECT * FROM books")

    result = mycursor.fetchall()
    mycursor.close()  # Zamknięcie kursora
    mydb.close()  # Zamknięcie połączenia
    return result


# Utworzenie widżetu Treeview
treeview = ttk.Treeview(root)
treeview["columns"] = ("id", "title", "author", "price", "category")
treeview.column("#0", width=0)
treeview.heading("id", text="ID")
treeview.heading("title", text="Tytuł")
treeview.heading("author", text="Autor")
treeview.heading("price", text="Cena")
treeview.heading("category", text="Kategoria")

# Wyświetlenie widżetu Treeview
treeview.pack()


def load_data():
    data = fetch_data()
    # Usunięcie poprzednich danych
    treeview.delete(*treeview.get_children())
    # Dodanie nowych danych
    for row in data:
        treeview.insert("", "end", values=(row[0], row[1], row[2], row[3], row[4]))

# Wywołanie funkcji wczytującej dane
load_data()

root.mainloop()
```



Aby dodać nową książkę utworzymy nowe okno za pomocą funkcji:

```
def open_new_book_window():
    new_window = tk.Toplevel(root)
    new_window.title("Dodaj nową książkę")

    title_label = ttk.Label(new_window, text="Tytuł:")
    title_label.pack()
    title_entry = ttk.Entry(new_window)
    title_entry.pack()

    author_label = ttk.Label(new_window, text="Autor:")
    author_label.pack()
    author_entry = ttk.Entry(new_window)
    author_entry.pack()

    price_label = ttk.Label(new_window, text="Cena:")
    price_label.pack()
    price_entry = ttk.Entry(new_window)
    price_entry.pack()

    category_label = ttk.Label(new_window, text="Kategoria:")
    category_label.pack()
    category_entry = ttk.Entry(new_window)
    category_entry.pack()
    def add_new():
        pass
    add_button = ttk.Button(new_window, text="Dodaj", command=add_new)
    add_button.pack()
```

i przycisku:

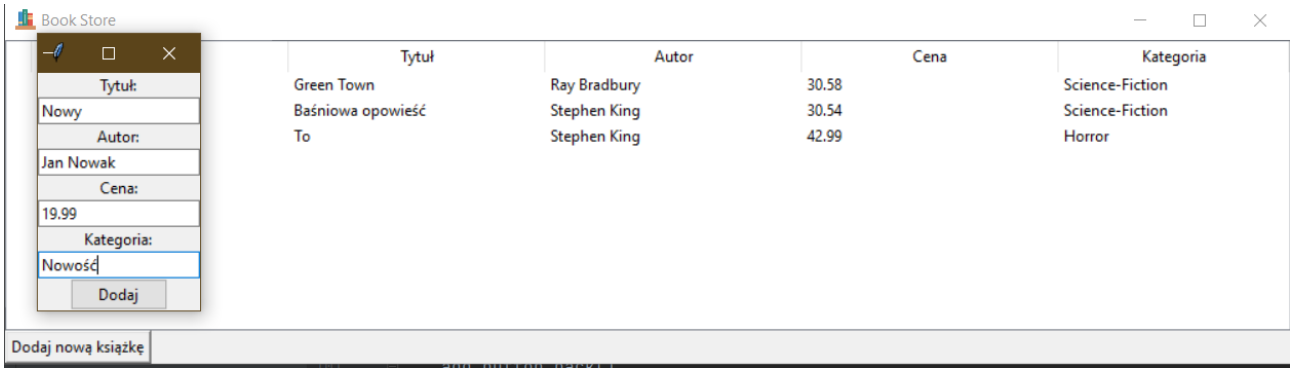
```
add_new_book_button = tk.Button(root, text="Dodaj nową książkę", command=open_new_book_window)
add_new_book_button.pack(side="left")
```

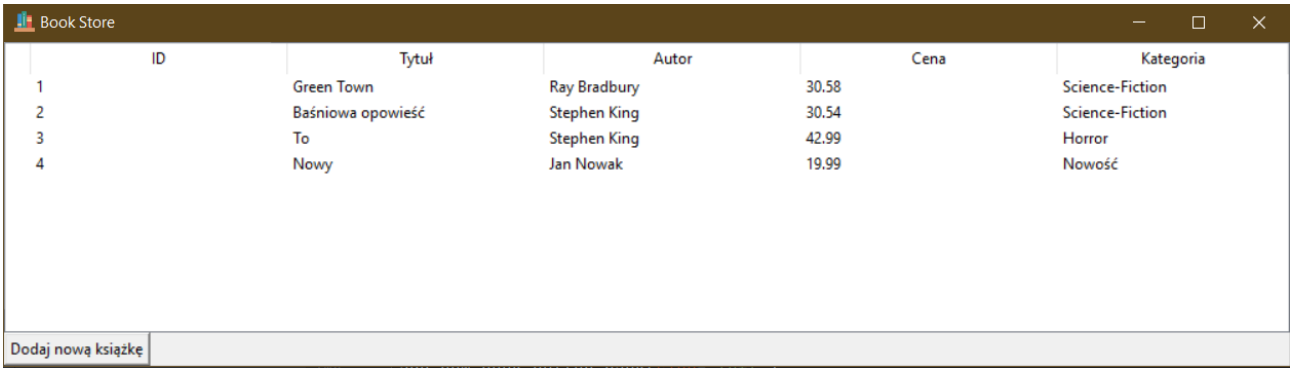
Jak widać w funkcji open_new_book_window() tworzymy też nowy przycisk add_button. Musimy też napisać funkcję, która wykona się po naciśnięciu tego przycisku – czyli wysłania danych dotyczących nowej książki, dane te pobierzemy z widżetów utworzonych w funkcji open_new_book_window().

Funkcja będzie miała postać:

```
def add_new():
    new_title = title_entry.get()
    new_author = author_entry.get()
    new_price = price_entry.get()
    new_category = category_entry.get()

    mydb = mysql.connector.connect(
        host="db4free.net",
        user="username",
        password="password",
        database="database"
    )
    mycursor = mydb.cursor()
    sql = "INSERT INTO books (title, author, price, category) VALUES (%s, %s, %s, %s)"

    params = (new_title, new_author, new_price, new_category)
    mycursor.execute(sql, params)
    mydb.commit() # Zapisanie zmian w bazie
    mycursor.close()  # Zamknięcie kursora
    mydb.close()  # Zamknięcie połączenia

    load_data()
    # Zamknięcie okna po zapisaniu zmian
    new_window.destroy()
```

Zapytanie SQL wstawia nowy rekord. W params wpisujemy krotkę z elementami do wstawienia, pobranymi z widżetów entry. %s jest znacznikiem zastępczym dla wartości, które zostaną podstawione.





Aby edytować i usuwać książki z treeview, należy podpiąć funkcję pod akcję – np. podwójne kliknięcie na element z treeview:

```
def open_details_window(event):
    # Pobranie zaznaczonego elementu
    selected_item = treeview.focus()
```

```
treeview.bind("<Double-1>",open_details_window)
```

W funkcji tak jak poprzednio powinniśmy utworzyć nowe okno, tym razem wczytując dane z zaznaczonego elementu:

```
def open_details_window(event):
    # Pobranie zaznaczonego elementu
    selected_item = treeview.focus()

    if selected_item:
        # Pobranie danych z  zaznaczonego elementu
        item_data = treeview.item(selected_item)
        item_values = item_data["values"]

        # Tworzenie nowego okna
        details_window = tk.Toplevel(root)
        details_window.title("Szczegóły")

        # Tworzenie i wyświetlanie widgetów opartych na danych z zaznaczonego elementu

        id_label = ttk.Label(details_window, text="ID:")
        id_label.pack()
        id_entry = ttk.Entry(details_window)
        id_entry.insert(0, item_values[0])
        id_entry.config(state="disabled") #Uniemożliwienie zmiany id
        id_entry.pack()

        title_label = ttk.Label(details_window, text="Tytuł:")
        title_label.pack()
        title_entry = ttk.Entry(details_window)
        title_entry.insert(0, item_values[1])
        title_entry.pack()

        author_label = ttk.Label(details_window, text="Autor:")
        author_label.pack()
        author_entry = ttk.Entry(details_window)
        author_entry.insert(0, item_values[2])
        author_entry.pack()

        price_label = ttk.Label(details_window, text="Cena:")
        price_label.pack()
        price_entry = ttk.Entry(details_window)
        price_entry.insert(0, item_values[3])
        price_entry.pack()

        category_label = ttk.Label(details_window, text="Kategoria:")
        category_label.pack()
        category_entry = ttk.Entry(details_window)
        category_entry.insert(0, item_values[4])
        category_entry.pack()
```

Następnie należałoby dodać 2 fuknkcje i 2 przyciski do modyfikowania rekordu i usuwania.

Przykład zapytania:

```
# Usuwanie:
sql = "DELETE FROM books WHERE id = %s"
params = (id_entry.get(),)
mycursor.execute(sql, params)
```

```
# Aktualizacja danych w bazie danych
sql = "UPDATE books SET title=%s, author=%s, price=%s, category=%s WHERE id=%s"
params = (new_title, new_author, new_price, new_category, same_id)
mycursor.execute(sql, params)
```

przy założeniu , że pod zmienne w params wstawiamy wartości pobrane z widgżetów.
Np: same_id = id_entry.get()

W SQLite, zapytania wyglądają podobnie:
Biblioteka jest wbudowana:
import sqlite3

```
def add_book(title, author, price, category):
    conn = sqlite3.connect('database.db')
    cursor = conn.cursor()
    cursor.execute("INSERT INTO Books (title, author, price, category) VALUES
(?, ?, ?, ?)",
                   (title, author, price, category))

    conn.commit()
    conn.close()
```

```
if __name__ == "__main__":
    conn = sqlite3.connect('database.db')
    cursor = conn.cursor()
    cursor.execute('''CREATE TABLE IF NOT EXISTS Books (
                        id INTEGER PRIMARY KEY AUTOINCREMENT,
                        title TEXT,
                        author TEXT,
                        price REAL,
                        category TEXT
                    )''')
    conn.commit()
    conn.close()
    create_book("database.db","Book 1", "Author 1", 9.99, "Fiction")
```

W Pythonie obsługa wyjątków odbywa się za pomocą bloku try-except. Blok try zawiera kod, który potencjalnie może zgłosić wyjątek, a blok except definiuje, jak obsłużyć zgłoszone wyjątki.

https://pl.wikibooks.org/wiki/Zanurkuj_w_Pythonie/Obs%C5%82uga_wyj%C4%85tk%C3%B3w

Jeśli w trakcie wykonywania kodu w try zgłoszony zostanie wyjątek, program przechodzi do bloku except, który odpowiada zgłoszonemu wyjątkowi.

Dodatkowo istnieją dwa opcjonalne bloki:
-else jest wykonywany tylko wtedy, gdy w bloku try nie został zgłoszony żaden wyjątek.

-finally jest wykonywany zawsze, niezależnie od tego, czy w bloku try został zgłoszony wyjątek czy nie. Jest przydatny do zamknięcia zasobów lub wykonania zadań, które muszą być wykonane niezależnie od wyjątku.

Prosty przykład z dzieleniem przez zero:

```
try:
    x = 10 / 0
except ZeroDivisionError:
    print("Dzielenie przez zero!")
except Exception as e:
    print(f"Wystąpił wyjątek: {str(e)}")
else:
    print("Operacja wykonana poprawnie.")
finally:
    print("Nareszcie !!")
```

Przykład funkcji dodawania książki z obsługą wyjątku:

```
def create_book(title, author, price, category):
    try:
        conn = sqlite3.connect('database.db')
        cursor = conn.cursor()
        cursor.execute("INSERT INTO Books (title, author, price, category)
VALUES (?, ?, ?, ?)",
                       (title, author, price, category))
        conn.commit()
    except sqlite3.Error as e:
        print(f"Error: {e}")
    finally:
        if cursor:
            cursor.close()
        if conn:
            conn.close()
```

Jeżeli np. tabelka nie istnieje (zamiast Books wpiszemy Books2) próbując wstawić książkę- dostaniemy błąd przechwycimy i wyświetlimy go:
Error: no such table: Books2

Zadanie:
Napisz aplikację okienkową pozwalającą wyświetlić(20%), zapisać(20%), usunąć(20%) oraz edytować(20%) dane dotyczące studentów (dane dotyczą ocen z przedmiotu – jak w poprzednim zadaniu). Dane mają być przechowywane w bazie danych SQLite.
Dodaj obsługę wyjątków przy połączeniach z bazą(20%).