

# Aplikacja sklep rowerowy

Została tak nazwana, ponieważ dobrze sprawdziłaby się w małym sklepie rowerowym.

Za jej pomocą można z łatwością sprawdzić jakie części i inne komponenty znajdują się na magazynie naszego sklepu.

Po załączeniu głównego okna aplikacji widzimy 4 zakładki, z którymi możemy prowadzić pełną interakcję, po otwarciu którejś z zakładek ukazuje się spis części, które aktualnie znajdują się w naszej bazie danych.

Niestety jest to dość podstawowa wersja tej aplikacji więc nie ma możliwości dodania lub usunięcia rekordu do danej kategorii z poziomu aplikacji, jedynie modyfikując kod „baza\_danych.py”.

## Oto opis strukturalny bazy danych (diagram ERD):



Każda z tabel ma inaczej zatytułowane kolumny.

## Oto kod SQL tworzący bazę danych:

```
import sqlite3

# Utwórz połączenie z bazą danych
conn = sqlite3.connect('baza_danych.db')

# Utwórz kursor do wykonywania poleceń SQL
cursor = conn.cursor()

# Utwórz tabelę "czesci"
cursor.execute("""CREATE TABLE IF NOT EXISTS czesci (
    id INTEGER PRIMARY KEY,
    nazwa TEXT,
    producent TEXT,
    cena REAL,
    ilosc INTEGER
)""")

# Dodaj dane do tabeli "czesci"
dane_czesci = [
    (1, 'Opona', 'Schwalbe', 50.00, 10),
    (2, 'Siodełko', 'Brooks', 100.00, 5),
    (3, 'Kierownica', 'Ritchey', 80.00, 8),
    (4, 'Przerzutka', 'Shimano', 120.00, 4),
    (5, 'Hamulce', 'Magura', 150.00, 6)
]

cursor.executemany('INSERT INTO czesci VALUES (?, ?, ?, ?, ?)', dane_czesci)

# Utwórz tabelę "ramy"
cursor.execute("""CREATE TABLE IF NOT EXISTS ramy (
    id INTEGER PRIMARY KEY,
    nazwa TEXT,
    producent TEXT,
    materiał TEXT,
    cena REAL
)""")
```

```
# Dodaj dane do tabeli "ramy"
```

```
dane_ramy = [
```

```
    (1, 'Rama MTB', 'Trek', 'Aluminiowa', 500.00),  
    (2, 'Rama szosowa', 'Specialized', 'Węglowa', 1000.00),  
    (3, 'Rama miejska', 'Giant', 'Stalowa', 300.00),  
    (4, 'Rama BMX', 'Redline', 'Aluminiowa', 400.00),  
    (5, 'Rama trekkingowa', 'Scott', 'Aluminiowa', 600.00)
```

```
]
```

```
cursor.executemany('INSERT INTO ramy VALUES (?, ?, ?, ?, ?)', dane_ramy)
```

```
# Utwórz tabelę "kola"
```

```
cursor.execute("""CREATE TABLE IF NOT EXISTS kola (
```

```
    id INTEGER PRIMARY KEY,
```

```
    nazwa TEXT,
```

```
    producent TEXT,
```

```
    rozmiar INTEGER,
```

```
    cena REAL
```

```
))""")
```

```
# Dodaj dane do tabeli "kola"
```

```
dane_kola = [
```

```
    (1, 'Koło MTB', 'Mavic', 26, 200.00),  
    (2, 'Koło szosowe', 'Zipp', 700, 800.00),  
    (3, 'Koło miejskie', 'Shimano', 28, 100.00),  
    (4, 'Koło BMX', 'Odyssey', 20, 150.00),  
    (5, 'Koło trekkingowe', 'Alexrims', 28, 180.00)
```

```
]
```

```
cursor.executemany('INSERT INTO kola VALUES (?, ?, ?, ?, ?)', dane_kola)
```

```
# Utwórz tabelę "akcesoria"
```

```
cursor.execute("""CREATE TABLE IF NOT EXISTS akcesoria (
```

```
    id INTEGER PRIMARY KEY,
```

```
    nazwa TEXT,
```

```
    producent TEXT,
```

```

        cena REAL,

        ilosc INTEGER

    )'''

# Dodaj dane do tabeli "akcesoria"
dane_akcesoria = [

    (1, 'Licznik rowerowy', 'Cateye', 30.00, 7),

    (2, 'Bidon', 'Elite', 10.00, 10),

    (3, 'Lampa przednia', 'Knog', 40.00, 6),

    (4, 'Zapięcie rowerowe', 'Abus', 20.00, 8),

    (5, 'Kask', 'Giro', 100.00, 4)

]

cursor.executemany('INSERT INTO akcesoria VALUES (?, ?, ?, ?, ?)', dane_akcesoria)

# Zatwierdź zmiany w bazie danych
conn.commit()


# Zamknij połączenie z bazą danych
conn.close()

```

**Baza już jest częściowo wypełniona danymi, jednak poniżej znajduje się kod, który wprowadza kolejne dane do bazy:**

```

import sqlite3

# Utwórz połączenie z bazą danych
conn = sqlite3.connect('baza_danych.db')

# Utwórz kursor do wykonywania poleceń SQL
cursor = conn.cursor()

# Wypełnij tabelę "czesci" przykładowymi danymi
dane_czesci = [

    (6, 'Pedal', 'Shimano', 50.00, 3),

    (7, 'Korba', 'SRAM', 120.00, 2),

    (8, 'Kaseta', 'Shimano', 80.00, 4),

    (9, 'Kierownica', 'Truvativ', 200.00, 5),

```

```

        (10, 'Widelec', 'RockShox', 300.00, 1)
    ]
    cursor.executemany('INSERT INTO czesci VALUES (?, ?, ?, ?, ?)', dane_czesci)
    # Wypełnij tabelę "ramy" przykładowymi danymi
    dane_ramy = [
        (6, 'Rama górską', 'Cannondale', 'Aluminiowa', 600.00),
        (7, 'Rama szosowa aero', 'Cervelo', 'Węglowa', 1500.00),
        (8, 'Rama miejska damska', 'Pashley', 'Stalowa', 400.00),
        (9, 'Rama dirt', 'NS Bikes', 'Aluminiowa', 500.00),
        (10, 'Rama trekkingowa damska', 'Kalkhoff', 'Aluminiowa', 700.00)
    ]
    cursor.executemany('INSERT INTO ramy VALUES (?, ?, ?, ?, ?)', dane_ramy)
    # Wypełnij tabelę "kola" przykładowymi danymi
    dane_kola = [
        (6, 'Koło górskie', 'Hope', 29, 400.00),
        (7, 'Koło aero', 'ENVE', 700, 1200.00),
        (8, 'Koło miejskie', 'Shimano', 28, 150.00),
        (9, 'Koło BMX Street', 'BSD', 20, 200.00),
        (10, 'Koło trekkingowe', 'Ryde', 28, 250.00)
    ]
    cursor.executemany('INSERT INTO kola VALUES (?, ?, ?, ?, ?)', dane_kola)
    # Wypełnij tabelę "akcesoria" przykładowymi danymi
    dane_akcesoria = [
        (6, 'Pompka', 'Topeak', 30.00, 5),
        (7, 'Kask dziecięcy', 'Bell', 40.00, 3),
        (8, 'Koszyk', 'Basil', 25.00, 7),
        (9, 'Dzwonek', 'Crane', 10.00, 10),
        (10, 'Bagażnik', 'Thule', 80.00, 2)
    ]
    cursor.executemany('INSERT INTO akcesoria VALUES (?, ?, ?, ?, ?)', dane_akcesoria)

```

```
# Zatwierdź zmiany w bazie danych
conn.commit()

# Zamknij połączenie z bazą danych
conn.close()
```

**Poniżej znajduje się kod źródłowy do aplikacji, która będzie korzystać z powyższej bazy danych:**

```
import tkinter as tk

import sqlite3

# Połączenie z bazą danych

conn = sqlite3.connect("baza_danych.db")
cursor = conn.cursor()

class Aplikacja:

    def __init__(self, master):

        self.master = master

        self.master.title("Sklep rowerowy")
        self.master.geometry("400x300")

# Ustawienie koloru tła

self.master.configure(bg="lightgray")

# Przyciski do wyświetlania danych

ramy_button = tk.Button(
    self.master,
    text="Wyświetl Ramy",
    command=self.wyświetl_ramy,
    bg="blue",
    fg="white",
)

ramy_button.pack(pady=10)
```

```
kola_button = tk.Button(  
    self.master,  
    text="Wyświetl Koła",  
    command=self.wyświetl_kola,  
    bg="green",  
    fg="white",  
)
```

```
kola_button.pack(pady=10)
```

```
akcesoria_button = tk.Button(  
    self.master,  
    text="Wyświetl Akcesoria",  
    command=self.wyświetl_akcesoria,  
    bg="orange",  
    fg="white",  
)
```

```
akcesoria_button.pack(pady=10)
```

```
czesci_button = tk.Button(  
    self.master,  
    text="Wyświetl Części",  
    command=self.wyświetl_czesci,  
    bg="purple",  
    fg="white",  
)
```

```
czesci_button.pack(pady=10)
```

```
def wyświetl_ramy(self):
```

```
    # Utworzenie nowego okna dla kategorii "Ramy"
```

```
    okno_ramy = tk.Toplevel(self.master)
```

```
okno_ramy.title("Ramy")  
okno_ramy.geometry("400x300")
```

#### **# Ustawienie koloru tła**

```
okno_ramy.configure(bg="lightblue")
```

#### **# Pobranie danych z tabeli "ramy"**

```
cursor.execute("SELECT * FROM ramy")  
dane = cursor.fetchall()
```

#### **# Wyświetlenie danych w listboxie**

```
listbox = tk.Listbox(okno_ramy)  
listbox.pack(side="left", fill="both", expand=True)
```

#### **# Dodanie paska przewijania dla listboxa**

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

#### **# Przypisanie paska przewijania do listboxa**

```
listbox.config(yscrollcommand=scrollbar.set)  
scrollbar.config(command=listbox.yview)
```

#### **# Wyświetlenie danych w listboxie**

```
for rama in dane:  
    listbox.insert("end", rama)
```

```
def wyswietl_kola(self):
```

#### **# Utworzenie nowego okna dla kategorii "Koła"**

```
okno_kola = tk.Toplevel(self.master)  
okno_kola.title("Koła")  
okno_kola.geometry("400x300")
```



#### **# Ustawienie koloru tła**

```
okno_kola.configure(bg="lightgreen")
```

#### **# Pobranie danych z tabeli "kola"**

```
cursor.execute("SELECT * FROM kola")
```

```
dane = cursor.fetchall()
```

#### **# Wyświetlenie danych w listboxie**

```
listbox = tk.Listbox(okno_kola)
```

```
listbox.pack(side="left", fill="both", expand=True)
```

#### **# Dodanie paska przewijania dla listboxa**

```
scrollbar = tk.Scrollbar(okno_kola)
```

```
scrollbar.pack(side="right", fill="y")
```

#### **# Przypisanie paska przewijania do listboxa**

```
listbox.config(yscrollcommand=scrollbar.set)
```

```
scrollbar.config(command=listbox.yview)
```

#### **# Wyświetlenie danych w listboxie**

```
for kolo in dane:
```

```
    listBox.insert("end", kolo)
```

```
def wyswietl_akcesoria(self):
```

#### **# Utworzenie nowego okna dla kategorii "Akcesoria"**

```
okno_akcesoria = tk.Toplevel(self.master)
```

```
okno_akcesoria.title("Akcesoria")
```

```
okno_akcesoria.geometry("400x300")
```

#### **# Ustawienie koloru tła**

```
okno_akcesoria.configure(bg="lightyellow")
```

**# Pobranie danych z tabeli "akcesoria"**

```
cursor.execute("SELECT * FROM akcesoria")
```

```
dane = cursor.fetchall()
```

**# Wyświetlenie danych w listboxie**

```
listbox = tk.Listbox(okno_akcesoria)
```

```
listbox.pack(side="left", fill="both", expand=True)
```

**# Dodanie paska przewijania dla listboxa**

```
scrollbar = tk.Scrollbar(okno_akcesoria)
```

```
scrollbar.pack(side="right", fill="y")
```

**# Przypisanie paska przewijania do listboxa**

```
listbox.config(yscrollcommand=scrollbar.set)
```

```
scrollbar.config(command=listbox.yview)
```

**# Wyświetlenie danych w listboxie**

```
for akcesorium in dane:
```

```
    listBox.insert("end", akcesorium)
```

```
def wyswietl_czesci(self):
```

**# Utworzenie nowego okna dla kategorii "Części"**

```
okno_czesci = tk.Toplevel(self.master)
```

```
okno_czesci.title("Części")
```

```
okno_czesci.geometry("400x300")
```

**# Ustawienie koloru tła**

```
okno_czesci.configure(bg="pink")
```

**# Pobranie danych z tabeli "czesci"**

```
cursor.execute("SELECT * FROM czesci")  
  
dane = cursor.fetchall()
```

#### **# Wyświetlenie danych w listboxie**

```
listbox = tk.Listbox(okno_czesci)  
  
listbox.pack(side="left", fill="both", expand=True)
```

#### **# Dodanie paska przewijania dla listboxa**

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

#### **# Przypisanie paska przewijania do listboxa**

```
listbox.config(yscrollcommand=scrollbar.set)  
  
scrollbar.config(command=listbox.yview)
```

#### **# Wyświetlenie danych w listboxie**

```
for czesc in dane:  
  
    listBox.insert("end", czesc)
```

#### **# Utworzenie głównego okna aplikacji**

```
root = tk.Tk()
```

#### **# Utworzenie obiektu klasy Aplikacja**

```
app = Aplikacja(root)
```

#### **# Uruchomienie pętli głównej aplikacji**

```
root.mainloop()
```

## Poniżej znajdują się 2 testy jednostkowe – jeden dla kodu bazy danych oraz jeden dla kodu aplikacji:

### TEST 1 – baza danych

```
import sqlite3
```

```
import unittest
```

```
class TestBazaDanych(unittest.TestCase):
```

```
    def setUp(self):
```

```
        self.conn = sqlite3.connect(':memory:')
```

```
        self.cursor = self.conn.cursor()
```

```
        self.cursor.execute("""CREATE TABLE IF NOT EXISTS czesci (
```

```
            id INTEGER PRIMARY KEY,
```

```
            nazwa TEXT,
```

```
            producent TEXT,
```

```
            cena REAL,
```

```
            ilosc INTEGER
```

```
        )""")
```

```
    def tearDown(self):
```

```
        self.cursor.execute('DROP TABLE czesci')
```

```
        self.conn.close()
```

```
    def test_dodaj_czesc(self):
```

```
        # Sprawdź dodawanie jednego rekordu do tabeli "czesci"
```

```
        dane_czesci = (1, 'Opona', 'Schwalbe', 50.00, 10)
```

```
        self.cursor.execute('INSERT INTO czesci VALUES (?, ?, ?, ?, ?)', dane_czesci)
```

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

```
        self.cursor.execute('SELECT COUNT(*) FROM czesci')
```

```
        result = self.cursor.fetchone()[0]
```

```
        self.assertEqual(result, 1)
```

```

def test_dodaj_wiele_czesci(self):
    # Sprawdź dodawanie wielu rekordów do tabeli "czesci"
    dane_czesci = [
        (1, 'Opona', 'Schwalbe', 50.00, 10),
        (2, 'Siodełko', 'Brooks', 100.00, 5),
        (3, 'Kierownica', 'Ritchey', 80.00, 8),
        (4, 'Przerzutka', 'Shimano', 120.00, 4),
        (5, 'Hamulce', 'Magura', 150.00, 6)
    ]

    self.cursor.executemany('INSERT INTO czesci VALUES (?, ?, ?, ?, ?)', dane_czesci)
    self.conn.commit()

    self.cursor.execute('SELECT COUNT(*) FROM czesci')
    result = self.cursor.fetchone()[0]
    self.assertEqual(result, 5)

if __name__ == '__main__':
    unittest.main()

```

W powyższym przykładzie przedstawiono testy jednostkowe dla funkcji 'test\_dodaj\_czesc' oraz 'test\_dodaj\_wiele\_czesci', które sprawdzają poprawność dodawania rekordów do tabeli "czesci"

Przykładowe testy wykonują się na bazie danych w pamięci (':memory:'), ale można je dostosować do działania na pliku bazy danych.

W testach wykorzystano moduł 'unittest', który jest wbudowany w Pythona. Testy jednostkowe pozwalają na automatyczne sprawdzanie poprawności działania kodu i są istotne w procesie tworzenia aplikacji.

## TEST 2 – aplikacja

```
import tkinter_test as tkt
```

```
import unittest
```

```
import sqlite3
```

```
class TestAplikacja(unittest.TestCase):
```

```
    def setUp(self):
```

```
        # Utwórz bazę danych i dodaj dane testowe
```

```
        self.conn = sqlite3.connect(":memory:")
```

```
        self.cursor = self.conn.cursor()
```

```
        self.cursor.execute("""CREATE TABLE IF NOT EXISTS czesci (
```

```
            id INTEGER PRIMARY KEY,
```

```
            nazwa TEXT,
```

```
            producent TEXT,
```

```
            cena REAL,
```

```
            ilosc INTEGER
```

```
        )""")
```

```
        dane_czesci = [
```

```
            (1, 'Opona', 'Schwalbe', 50.00, 10),
```

```
            (2, 'Siodełko', 'Brooks', 100.00, 5),
```

```
            (3, 'Kierownica', 'Ritchey', 80.00, 8),
```

```
        ]
```

```
        self.cursor.executemany('INSERT INTO czesci VALUES (?, ?, ?, ?, ?)', dane_czesci)
```

```
        self.root = tkt.Tk()
```

```
        self.app = Aplikacja(self.root)
```

```
    def tearDown(self):
```

```
        # Usuń tabelę z bazą danych i zamknij połączenie
```

```

self.cursor.execute("DROP TABLE IF EXISTS czesci")

self.conn.close()

# Zniszcz obiekt aplikacji

self.app = None

# Zamknij główne okno aplikacji

self.root.destroy()

def test_wyswietl_czesci(self):

    # Wywołaj metodę wyswietl_czesci()

    self.app.wyswietl_czesci()

    # Sprawdź, czy wyświetlone dane są poprawne

    okno_czesci = self.app.okno_czesci

    listbox = okno_czesci.children['!listbox']

    items = listbox.get(0, 'end')

    self.assertEqual(len(items), 3) # Oczekujemy 3 elementów w liście

    self.assertIn('Opona', items) # Oczekujemy, że 'Opona' jest w liście

if __name__ == '__main__':

    unittest.main()

```

W tym przykładzie testowym utworzono tymczasową bazę danych w pamięci za pomocą `sqlite3.connect(":memory:")`. Dodatkowo, utworzono testową tabelę "czesci" i dodano kilka wierszy danych. Następnie, w funkcji `setUp()`, utworzono instancję aplikacji `Aplikacja` i główne okno `Tk` dla testów.

W teście `test_wyswietl_czesci()` wywołano metodę `wyswietl_czesci()` aplikacji, a następnie sprawdzono, czy dane wyświetlane w liście są poprawne.

## Dokumentacja pydoc (HTML)

[index](#)

sklep\_rowerowy [c:\users\ficek\desktop\sklep\\_rowerowy\aplikacja\sklep\\_rowerowy.py](#)

### Modules

[sqlite3](#) [tkinter](#)

### Classes

[builtins.object](#)

[Aplikacja](#)

class **Aplikacja**([builtins.object](#))

[Aplikacja](#)(master)

Methods defined here:

**\_\_init\_\_**(self, master)

Initialize self. See help(type(self)) for accurate signature.

wyswietl\_akcesoria(self)

wyswietl\_czesci(self)

wyswietl\_kola(self)

wyswietl\_ramy(self)

---

Data descriptors defined here:

**\_\_dict\_\_**

dictionary for instance variables (if defined)

**\_\_weakref\_\_**

list of weak references to the object (if defined)

### Data

**app** = <sklep\_rowerowy.Aplikacja object>

**conn** = <sqlite3.Connection object>

**cursor** = <sqlite3.Cursor object>

**root** = <tkinter.Tk object .>



## Instrukcja instalacji i uruchamiania aplikacji:

Aby uruchomić kod i zobaczyć aplikację w działaniu wykonaj:

1. Zapisz kod źródłowy aplikacji z rozszerzeniem .py.
2. Zapisz kod bazy danych również z rozszerzeniem .py.
3. Umieść oba kody w jednym katalogu.
4. Przy pomocy skrótu klawiszowego WINDOWS+R włącz się wiersz poleceń, komendą cd zmieniasz ścieżkę na tą, gdzie znajduje się twój katalog.
5. Komendą "python baza\_danych.py" włączasz bazę, powinien stworzyć się plik .db w katalogu.
6. Komendą "python sklep\_rowerowy.py" włączasz aplikację.

Aplikacja umożliwia tylko sprawdzenie, jaki sprzęt jest dostępny na magazynie sklepu rowerowego, klikając lewym przyciskiem myszy na zakładki „Wyświetl ramy, koła, części i akcesoria” możemy zobaczyć jakie produkty przypisaliśmy do jakiej kategorii.

### Wykonali:

Wiktor Fickowski

Krzysztof Krasnodębski

Marcel Malik