Kacper Grabalski 2ID12A
Krzysztof Giemza 2ID12A
Projekt Języki Skryptowe
Temat: Lokalna sieć wymiany książek

Dr inż. Dariusz Michalski. Formularz samooceny do projektu z języków skryptowych

Nr	Obszar	Wymaganie	KOD		Przyznane pkt	Pkt max
1	UI	JEST		☐		
		Wprowadzanie danych	user_interface.py Kod: def add_user(users, books): name = input("Imię i nazwisko: ") user_id = max([u.user_id for u in users], default=0) + 1 users.append(User(name, user_id)) save_data(users, books) print(f"Utworzono użytkownika: {name} (ID: {user_id})")	☐		2

		Wyświetlanie danych	user_interface.py
			Kod:
			def search_books(books):
			term = input("Szukaj (tytuł/autor/gatunek): ").lower()
			found_books = list(filter(lambda b: term in b.title.lower()
			or term in b.author.lower()
			or term in b.genre.lower(), books))
			
			if not found_books:
			print("Brak wyników!")
			return
			
			print("\nZnalezione książki:")
			for book in found_books:
			print(f"{book.title} - {book.author} ({book.genre})")
				☐		2

		Zmiana danych	user_interface.py kod: book_a.owner_id = user_b_id book_b.owner_id = user_a_id	☐		2

		Wyszukiwanie danych	user_interface.py
			Kod:
			def search_books(books):
			term = input("Szukaj (tytuł/autor/gatunek): ").lower()
			found_books = list(filter(lambda b: term in b.title.lower()
			or term in b.author.lower()
			or term in b.genre.lower(), books))
			
			if not found_books:
			print("Brak wyników!")
			return
			
			print("\nZnalezione książki:")
			for book in found_books:
			print(f"{book.title} - {book.author} ({book.genre})")
				☐		2

		Przedstawienie	visualization.py kod:	☐		2

		wyników	plt.bar(genre_count.keys(), genre_count.values()) plt.title("Rozkład książek według gatunków") plt.savefig("books_genre_chart.png")			
2	Podstawy	Zmienne	main.py kod: users, books = load_data()	☐		2
		typy danych	Stringi, inty, listy, słowniki np. lista w user_interface.py def display_stats(books): genres = [b.genre for b in books]	☐		2
		komentarze	main.py # Wczytanie danych przy starcie aplikacji	☐		1
		operatory	user_interface (operator +)) kod: book_id = max([b.book_id for b in books], default=0) + 1	☐		1,5

		Instrukcje warunkowe (if, elif, else)	user_interface.py
			kod:
			choice = input("Wybierz opcję: ")
			
			if choice == "1":
			add_user(users, books)
			elif choice == "2":
			add_book(users, books)
			elif choice == "3":
			search_books(books)
			elif choice == "4":
			exchange_book(users, books)
			elif choice == "5":
			display_stats(users, books)
			elif choice == "6":
			search_users(users)
			elif choice == "7":
			break
			else:
			print("Nieprawidłowy wybór!")
				☐		3

		Instrukcje iteracyjne				
		for	user_interface.py kod: for book in found_books: print(f"{book.title} - {book.author} ({book.genre})")	☐		2
		while	user_interface.py kod: while True: print("\n=== Lokalna Sieć Wymiany Książek ===") print("1. Dodaj użytkownika") print("2. Dodaj książkę") print("3. Wyszukaj książki") print("4. Wymień książkę") print("5. Wyświetl statystyki i użytkowników") print("6. Wyszukaj użytkownika") print("7. Wyjście") choice = input("Wybierz opcję: ")	☐		2
		Operacje wejścia (input)	user_interface.py kod: choice = input("Wybierz opcję: ")	☐		1,5

		Operacje wyjścia (print)	main.py kod; print("Dane zostały zapisane. Do widzenia!")	☐		1,5

		Funkcje z parametrami i wartościami zwracanymi	user_interface.py
			kod:
			def search_books(books):
			term = input("Szukaj (tytuł/autor/gatunek): ").lower()
			found_books = list(filter(lambda b: term in b.title.lower()
			or term in b.author.lower()
			or term in b.genre.lower(), books))
			
			if not found_books:
			print("Brak wyników!")
			return
			
			print("\nZnalezione książki:")
			for book in found_books:
			print(f"{book.title} - {book.author} ({book.genre})")
				☐		2

		Funkcje rekurencyjne	utils.py
			kod:
			def recursive_factorial(n):
			if n == 0:
			return 1
			return n * recursive_factorial(n-1)
				☐		3

		Funkcje przyjmujące inne funkcje jako argumenty	search_books.py (funkcja fillter) kod: def search_books(books): term = input("Szukaj (tytuł/autor/gatunek): ").lower() found_books = list(filter(lambda b: term in b.title.lower() or term in b.author.lower() or term in b.genre.lower(), books))	☐		3

		Dekoratory	utils.py
			kod:
			def log_activity(func):
			def wrapper(*args, **kwargs):
			print(f"Wywołano: {func.__name__}")
			return func(*args, **kwargs)
			return wrapper
				☐		1,5

3	Kontenery	Użycie listy	user.py kod: self.books_owned = books_owned if books_owned else []	☐		2
		Użycie słownika	user_interface.py kod: genre_count = {genre: genres.count(genre) for genre in unique_genres}	☐		2
		Użycie zbioru	user_interface.py kod: unique_genres = set(genres)	☐		1,5
		Użycie krotki	storage.py kod: return users, books	☐		1,5
4	Przestrzenie nazw	Zastosowano zmienne lokalne	user_interface.py kod: def add_user(users):	☐		1,5

			name = input("Imię i nazwisko: ") # Zmienna lokalna 'name'
			user_id = max([u.user_id for u in users], default=0) + 1 # Zmienna lokalna 'user_id'
			users.append(User(name, user_id))			

		Zastosowano zmienne globalne	storage.py kod: DATA_FILE = "library_data.json"	☐		1,5

		Zastosowano zakresy funkcji	utils.py (funkcja zagnieżdżona)
			kod:
			def log_activity(func):
			def wrapper(*args, **kwargs):
			print(f"Wywołano: {func.__name__}")
			return func(*args, **kwargs)
			return wrapper
				☐		1,5

		Zastosowano zakresy klas	user.py
			kod:
			class User:
			def __init__(self, name, user_id): # Metoda klasy
			self.name = name # Atrybut instancji
				☐		1,5

5	Moduły i pakiety	Projekt podzielony na moduły (import, __init__)		☐		2

Dr inż. Dariusz Michalski. Formularz samooceny do projektu z języków skryptowych

Nr	Obszar	Wymaganie	KOD		Przyznane pkt	Pkt max
		Własne pakiety/funkcje pomocnicze w osobnych plikach .py	Struktura plików: main.py storage.py user_interface.py user.py book.py utils.py visualization.py	☐		2
6	Obsługa błędów	Obsługa wyjątków (try, except, finally)	storage.py kod: try: with open(DATA_FILE, 'r') as f: data = json.load(f) except (FileNotFoundError, json.JSONDecodeError): return [], []	☐		2
		Użycie assert do testów i walidacji	test_units.py kod: def test_book_creation(self): book = Book("Python Basics", "A. Smith", 1, "Programming", 101) self.assertEqual(book.title, "Python Basics")	☐		1,5
7	Łańcuchy znaków	Operacje na stringach (m.in. formatowanie, dzielenie,	user_interface.py kod: term = input("Szukaj (tytuł/autor/gatunek): ").lower() if term in b.title.lower() or term in b.author.lower()	☐		2

		wyszukiwanie)				
8	Obsługa plików	Odczyt z plików .txt, .csv, .json, .xml (min. 1)	storage.py kod: # Zapis with open(DATA_FILE, 'w') as f: json.dump(data, f, indent=2) # Odczyt with open(DATA_FILE, 'r') as f: data = json.load(f)	☐		2

		Zapis do plików .txt, .csv, .json, .xml (min. 1)	storage.py
			kod:
			def save_data(users, books):
			"""Zapisuje dane do pliku JSON przed zamknięciem aplikacji"""
			data = {
			'users': [{
			'name': user.name,
			'user_id': user.user_id,
			'books_owned': user.books_owned
			} for user in users],
			
			'books': [{
			'title': book.title,
			'author': book.author,
			'book_id': book.book_id,
			'genre': book.genre,
			'owner_id': book.owner_id,
			'status': book.status
			} for book in books]
			}
			
			try:
			with open(DATA_FILE, 'w', encoding='utf-8') as f:
			json.dump(data, f, indent=2, ensure_ascii=False)
			except IOError as e:
			print(f"Błąd zapisu danych: {e}")
				☐		2

9	OOP	Klasy	user.py
			kod:
			class User:
			def __init__(self, name, user_id, books_owned=None):
			self.name = name
			self.user_id = user_id
			self.books_owned = books_owned if books_owned else []
			
			def add_book(self, book_id):
			if book_id not in self.books_owned:
			self.books_owned.append(book_id)
			
			def remove_book(self, book_id):
			if book_id in self.books_owned:
			self.books_owned.remove(book_id)
				☐		2

		Metody	user.py
			kod:
			def add_book(self, book_id):
			if book_id not in self.books_owned:
			self.books_owned.append(book_id)
				☐		2

		Konstruktory	book.py kod: def __init__(self, title, author, book_id, genre, owner_id): self.title = title self.author = author self.book_id = book_id self.genre = genre	☐		2

			self.owner_id = owner_id self.status = "available"			
		Dziedziczenie	-	☐		2
10	Programowanie funkcyjne	map	-	☐		1,5
		filter	user_interface.py kod: found_books = list(filter(lambda b: term in b.title.lower(), books))	☐		1,5
		lambda	user_interface.py kod: list(filter(lambda b: term in b.title.lower(), books))	☐		1,5

		reduce	storage.py
			kod:
			from functools import reduce
			# Przykładowe użycie reduce do sumowania
			total_books = reduce(lambda x, y: x + len(y.books_owned), users, 0)
				☐		1,5

11	Wizualizacja danych	Wygenerowano wykres (np. matplotlib, seaborn)	visualization.py kod: plt.bar(genre_count.keys(), genre_count.values()) plt.savefig("books_genre_chart.png")	☐		2
		Zapisano wykres do pliku graficznego (.png lub .jpg)	visualization.py kod: plt.bar(genre_count.keys(), genre_count.values()) plt.savefig("books_genre_chart.png")	☐		1,5

T12	Testowanie	Testy jednostkowe (assert, unittest, pytest)	test_units.py
			kod:
			class TestLibrary(unittest.TestCase):
			  def test_book_creation(self):
			  book = Book("Python Basics", "A. Smith", 1, "Programming", 101)
			  self.assertEqual(book.title, "Python Basics")
				☐		1,5

		Testy funkcjonalne	test_functional.py
			kod:
			# test_functional.py
			_import unittest
			from unittest.mock import patch, MagicMock
			import main # Importujemy główny moduł
			
			class TestFunctional(unittest.TestCase):
			  @patch('builtins.input', side_effect=['7'])
			  def test_main_menu_exit(self, mock_input):
			  # Utwórz mocki dla zależności
			  mock_load = MagicMock(return_value=([], []))
			  mock_save = MagicMock()
			  mock_chart = MagicMock()
			
			  # Podmień oryginalne funkcje mockami
			  with patch('main.load_data', mock_load), \
			  patch('main.save_data', mock_save), \
			
			patch('main.generate_genre_chart', mock_chart):
				☐		1,5

			# Uruchom główną funkcję
			main.main()
			
			# Sprawdź wywołania
			mock_load.assert_called_once()
			
			mock_save.assert_called_once_with([], [])
			
			mock_chart.assert_called_once_with([])
			
			if __name__ == "__main__":
			unittest.main()			

		Testy Integracyjne	test_integration.py
			kod:
			# test_integration.py
			import unittest
			import os
			import json
			from main import main
			from storage import load_data, save_data
			from user import User
			from book import Book
			from unittest.mock import patch
			
			class TestIntegration(unittest.TestCase):
			TEST_DATA_FILE =
			"test_library_data.json"
			
			def setUp(self):
			# Ustaw plik testowy
			self.original_data_file =
			"../../Desktop/projekt_skryptowe/Jezyki_Skryptowe_Proj_2ID12A_Kacper_Grabalski_Krzysztof_Giemza/kodziki/library_data.json"
			import storage
			storage.DATA_FILE =
			self.TEST_DATA_FILE
			
			# Usuń plik testowy jeśli istnieje
			if os.path.exists(self.TEST_DATA_FILE):
			os.remove(self.TEST_DATA_FILE)
			
			def tearDown(self):
			# Przywróć oryginalny plik danych
			import storage
			storage.DATA_FILE =
			self.original_data_file
			# Usuń plik testowy
			if os.path.exists(self.TEST_DATA_FILE):
			os.remove(self.TEST_DATA_FILE)
			
			def test_full_flow(self):
			# Symuluj interakcję użytkownika
			inputs = [
			'1', # Dodaj użytkownika
			'Jan Kowalski',
			'2', # Dodaj książkę
			'1', # ID właściciela
			'Python dla poczatkujacych',
			'Marek Nowak',
			'Programowanie',
			'2', # Dodaj kolejną książkę
			'1', # ID właściciela
			'Python zaawansowany',
			'Jan Nowak',
			'Programowanie',
			'5', # Wyświetl statystyki
			'7' # Wyjście
			]
			
			with patch('builtins.input',
			side_effect=inputs):
			main()
				☐		1,5

			# Sprawdź dane w pliku
			with open(self.TEST_DATA_FILE, 'r') as f:
			data = json.load(f)
			
			# Sprawdź użytkowników
			self.assertEqual(len(data['users']), 1)
			
			self.assertEqual(data['users'][0]['name'], 'Jan Kowalski')
			
			# Sprawdź książki
			self.assertEqual(len(data['books']), 2)
			titles = [book['title'] for book in data['books']]
			self.assertIn('Python dla poczatkujacych', titles)
			self.assertIn('Python zaawansowany', titles)
			
			def test_exchange_flow(self):
			# Przygotuj dane testowe
			user1 = User("Jan Kowalski", 1)
			user2 = User("Anna Nowak", 2)
			book1 = Book("Ksiazka A", "Autor A", 101, "Gatunek A", 1)
			book2 = Book("Ksiazka B", "Autor B", 102, "Gatunek B", 2)
			
			save_data([user1, user2], [book1, book2])
			
			# Symuluj wymianę
			inputs = [
			'4', # Wymień książkę
			'1', # ID użytkownika A
			'101', # ID książki do oddania
			'2', # ID użytkownika B
			'102', # ID książki do otrzymania
			'7' # Wyjście
			]
			
			with patch('builtins.input', side_effect=inputs):
			main()
			
			# Sprawdź dane po wymianie
			users, books = load_data()
			
			# Sprawdź właścicieli książek
			book1_owner = next((b.owner_id for b in books if b.book_id == 101), None)
			book2_owner = next((b.owner_id for b in books if b.book_id == 102), None)
			
			self.assertEqual(book1_owner, 2)
			self.assertEqual(book2_owner, 1)
			
			# Sprawdź listy książek użytkowników
			user1_books = next((u.books_owned for u in users if u.user_id == 1), [])
			user2_books = next((u.books_owned for u in users if u.user_id == 2), [])
			
			self.assertIn(102, user1_books)
			self.assertNotIn(101, user1_books)
			self.assertIn(101, user2_books)
			self.assertNotIn(102, user2_books)
			
			if __name__ == '__main__':
			unittest.main()			

		Testy graniczne / błędne dane	test_boundary.py kod: # test_boundary.py	☐		1,5

```
import unittest
import os
from storage import save_data, load_data
from user import User
from book import Book


class TestBoundary(unittest.TestCase):
    TEST_DATA_FILE =
"test_boundary_data.json"

    def setUp(self):
        import storage
        self.original_data_file =
storage.DATA_FILE
        storage.DATA_FILE =
self.TEST_DATA_FILE
        if os.path.exists(self.TEST_DATA_FILE):
            os.remove(self.TEST_DATA_FILE)

    def tearDown(self):
        import storage
        storage.DATA_FILE =
self.original_data_file
        if os.path.exists(self.TEST_DATA_FILE):
            os.remove(self.TEST_DATA_FILE)

    def test_large_data(self):
        # Tworzenie dużej ilośći danych
        num_users = 1000
        num_books = 10000

        users = []
        for i in range(num_users):
            users.append(User(f"User{i}", i))

        books = []
        for i in range(num_books):
            owner_id = i % num_users
            books.append(Book(f"Book{i}",
f"Author{i}", i, "Genre", owner_id))
            users[owner_id].add_book(i)

        # Zapis i odczyt
        save_data(users, books)
        loaded_users, loaded_books =
load_data()

        # Sprawdź poprawność
        self.assertEqual(len(loaded_users),
num_users)
        self.assertEqual(len(loaded_books),
num_books)

self.assertEqual(len(loaded_users[0].books_
owned), num_books // num_users)

    def test_special_characters(self):
        # Testowanie specjalnych znaków
        user = User("Zażółć gęślą jaźń", 1)
        book = Book("Książka z € & # @",
"Автор Їжак", 1, "Gatunek", 1)

        save_data([user], [book])
        loaded_users, loaded_books =
load_data()

        self.assertEqual(loaded_users[0].name,
"Zażółć gęślą jaźń")
        self.assertEqual(loaded_books[0].title,
"Książka z € & # @")

self.assertEqual(loaded_books[0].author,
"Автор Їжак")
```

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

		Testy wydajności (np. czas wykonania, timeit)	test_performance.py
			kod:
			# test_performance.py
			import timeit
			import os
			from storage import save_data, load_data
			from user import User
			from book import Book
			
			def setup_data(num_users=100, num_books=1000):
			  users = [User(f"User{i}", i) for i in range(num_users)]
			  books = [Book(f"Book{i}", f"Author{i}", i, "Genre", i % num_users) for i in range(num_books)]
			  for i, user in enumerate(users):
			  user.books_owned = [j for j in range(num_books) if j % num_users == i]
			  return users, books
			
			def test_save():
			  users, books = setup_data()
			  save_data(users, books)
			
			def test_load():
			  load_data()
			
			if __name__ == '__main__':
			  # Przygotuj dane
			  users, books = setup_data()
			  save_data(users, books)
			
			  # Test zapisu
			  save_time = timeit.timeit("test_save()", setup="from __main__ import test_save", number=10)
			  print(f"Średni czas zapisu: {save_time / 10:.4f}s")
			
			  # Test odczytu
			  load_time = timeit.timeit("test_load()", setup="from __main__ import test_load", number=10)
			  print(f"Średni czas odczytu: {load_time / 10:.4f}s")
			
			  # Test dużej ilości danych
			  users, books = setup_data(1000, 10000)
			  save_data(users, books)
			
			  large_load_time = timeit.timeit("test_load()", setup="from __main__ import test_load", number=5)
			  print(f"Średni czas odczytu dużej ilości danych: {large_load_time / 5:.4f}s")
				☐		1,5

		Testy pamięci memory_profiler	test_memory.py kod: from memory_profiler import profile import os import tempfile import shutil import storage from storage import save_data, load_data @profile	☐		1,5

			def memory_test():
			  # Utwórz tymczasowy katalog
			  test_dir = tempfile.mkdtemp()
			  test_file = os.path.join(test_dir, "memory_test_data.json")
			
			  try:
			  # Ustaw plik testowy
			  original_data_file = storage.DATA_FILE
			  storage.DATA_FILE = test_file
			
			  # Tworzenie danych
			  from user import User
			  from book import Book
			  num_users = 500
			  num_books = 5000
			  users = [User(f"User{i}", i) for i in range(num_users)]
			  books = [Book(f"Book{i}", f"Author{i}", i, "Genre", i % num_users) for i in range(num_books)]
			
			  # Dodawanie książek do użytkowników
			  for i, user in enumerate(users):
			  user.books_owned = [j for j in range(num_books) if j % num_users == i]
			
			  # Zapis i odczyt
			  save_data(users, books)
			  loaded_users, loaded_books = load_data()
			
			  # Czyszczenie
			  del loaded_users
			  del loaded_books
			
			  finally:
			  # Przywróć oryginalny plik danych
			  storage.DATA_FILE = original_data_file
			  # Usuń tymczasowy katalog
			  shutil.rmtree(test_dir)
			
			if __name__ == '__main__':
			  memory_test()			

		Test jakości kodu (flake8, pylint)	test_jak.py
			kod:
			import unittest
			import subprocess
			import sys
			
			class TestCodeQuality(unittest.TestCase):
			  def test_flake8(self):
			  """Test zgodności kodu z PEP8 używając flake8"""
			  try:
			  result = subprocess.run(
			  [sys.executable, '-m', 'flake8', '--exclude', 'venv,.venv', '--count', '--select=E9,F63,F7,F82',
			  '--show-source', '--statistics', '.'],
			  capture_output=True,
			  text=True
			  )
			
			  if result.returncode != 0:
			  print("Błędy flake8:")
			  print(result.stdout)
			  self.assertEqual(result.returncode, 0, "Znaleziono błędy w kodzie (flake8)")
			  except Exception as e:
			  self.skipTest(f"flake8 nie jest dostępny: {str(e)}")
				☐		1,5

			def test_pylint(self):
			"""Test jakości kodu używając pylint"""
			modules = ['main.py', 'storage.py', 'user_interface.py', 'user.py', 'book.py', 'utils.py', 'visualization.py']
			errors = []
			
			try:
			for module in modules:
			result = subprocess.run(
			[sys.executable, '-m', 'pylint', '--disable=R,C', module],
			capture_output=True,
			text=True
			)
			
			if result.returncode != 0:
			errors.append(f"Błędy w {module}:\n{result.stdout}")
			
			if errors:
			print("\n".join(errors))
			self.assertFalse(errors, "Znaleziono błędy w kodzie (pylint)")
			except Exception as e:
			self.skipTest(f"pylint nie jest dostępny: {str(e)}")
			
			if __name__ == '__main__':
			unittest.main()			

13	Wersjonowanie	Repozytorium GIT	https://github.com/DeskaV2/Projekt_Skryptowe.git	☐		1
		Historia commitów		☐		1

Dr inż. Dariusz Michalski. Formularz samooceny do projektu z języków skryptowych

Nr	Obszar	Wymaganie	KOD		Przyznane pkt	Pkt max
		Link do GitHub	https://github.com/DeskaV2/Projekt_Skryptowe.git	☐		1
		Opis commitów		☐		1
14	Dokumentacja	Plik README.md (cel, autorzy, uruchamianie)	Jest utworzony i może zostać otwarty po pobraniu repozytorium	☐		1,5
		Przykładowe dane wejściowe i wyjściowe	Wybierz opcję: 1 Imię i nazwisko: Jan Kowalski Utworzono użytkownika: Jan Kowalski (ID: 1)	☐		2
		Diagram klas lub struktura modułów	. ├── main.py # Główny punkt wejścia ├── storage.py # Obsługa danych (JSON) ├── user_interface.py # Interfejs użytkownika ├── user.py # Model użytkownika ├── book.py # Model książki ├── utils.py # Funkcje pomocnicze	☐		2

			├── visualization.py # Generowanie wykresów ├── test_unit.py # Testy jednostkowe ├── test_fun.py # Testy funkcjonalne ├── test_integ.py # Testy integracyjne ├── test_gran.py # Testy graniczne ├── test_wyda.py # Testy wydajności ├── test_pam.py # Testy pamięci ├── test_jak.py # Testy jakości └── library_data.json # Baza danych (automatycznie tworzona)			
		SUMA				