### Projekt Zaliczeniowy z Języków Skryptowych: Trener matematyczny

**Odtwarzacz Quizów Edukacyjnych**

**1. Strona Tytułowa**

* **Nazwa Uczelni:** Politechnika Świętokrzyska
* **Nazwa Przedmiotu:** Języki Skryptowe
* **Tytuł naukowy oraz imię i nazwisko prowadzącego zajęcia:** Dr inż. Dariusz Michalski
* **Tytuł Projektu:** Trener matematyczny – quiz i analiza postępów
* **Imiona i nazwiska członków zespołu:** Jakub Pietrzykowski, Piotr Pomarański
* **Grupa studencka:** 2ID13B
* **Data oddania:** 21.06.2025

**2. Cel projektu**

Głównym celem projektu "Trener Matematyczny" było stworzenie interaktywnej aplikacji konsolowej, która umożliwia użytkownikom ćwiczenie i rozwijanie swoich umiejętności matematycznych w różnych obszarach. Projekt ma na celu dostarczenie elastycznego i rozszerzalnego narzędzia do samodzielnej nauki, wspierającego poprawę kompetencji w zakresie arytmetyki, ułamków, potęg i innych, przyszłych dziedzin matematyki.

**3. Zakres funkcjonalny**

Aplikacja "Trener Matematyczny" oferuje następujące kluczowe funkcjonalności:

* **Wybór Quizu:** Użytkownik może wybrać jeden z dostępnych rodzajów quizów (np. "Podstawy Arytmetyki", "Quiz z Ułamków", "Potęgi").
* **Definiowanie Liczby Pytań:** Możliwość określenia liczby pytań w danej sesji quizowej, co pozwala na dostosowanie długości treningu.
* **Interaktywna Sesja Quizowa:** Prezentowanie pytań użytkownikowi i zbieranie jego odpowiedzi.
* **Walidacja Odpowiedzi:** Natychmiastowe sprawdzanie poprawności wprowadzonych odpowiedzi i informowanie o wyniku.
* **Zliczanie Wyników:** Sumowanie poprawnych odpowiedzi w trakcie sesji.
* **Śledzenie Postępów Użytkownika:** Zapisywanie wyników z ukończonych quizów, co umożliwia monitorowanie historii osiągnięć.
* **Wyświetlanie Postępów:** Prezentowanie zebranych danych o postępach, aby użytkownik mógł zobaczyć swoje osiągnięcia i obszary do poprawy.

**4. Struktura projektu**

Projekt został zorganizowany w przejrzyste pakiety i moduły, co zapewnia wysoką modularność i ułatwia zarządzanie kodem:

* main.py: Główny moduł uruchamiający aplikację. Koordynuje przepływ programu, prezentuje menu użytkownikowi i inicjuje sesje quizowe oraz zarządzanie postępami.
* core/: Pakiet zawierający podstawową logikę aplikacji.
  + quiz\_manager.py: Moduł odpowiedzialny za ładowanie definicji quizów (klas) oraz pytań z plików JSON. Zarządza dostępnymi quizami i przygotowuje sesje quizowe.
  + user\_progress.py: Moduł do zarządzania postępami użytkownika, w tym zapisywania wyników quizów do pliku i ich odczytywania.
  + utils.py: Moduł zawierający pomocnicze funkcje ogólnego przeznaczenia (np. get\_positive\_integer\_input do walidacji danych wejściowych).
* quizzes/: Pakiet zawierający definicje poszczególnych rodzajów quizów. Każdy quiz jest osobnym modułem (np. basic\_arithmetic.py, fraction\_quiz.py, powers.py), implementującym metody get\_name() i check\_answer().
* data/: Katalog przeznaczony na pliki danych, takie jak quiz\_data.json (zawierający pytania i odpowiedzi do quizów) oraz plik z postępami użytkownika.
* tests/: Pakiet zawierający testy jednostkowe i integracyjne dla różnych komponentów aplikacji, zapewniające jakość i poprawność działania kodu.
  + test\_core/: Testy dla modułów z pakietu core (np. test\_quiz\_manager.py).

**5. Technologie i biblioteki**

Projekt został zrealizowany przy użyciu następujących technologii:

* **Python 3.11**: Główny język programowania.
* **Standardowe Biblioteki Pythona**:
  + os: Do operacji na ścieżkach i katalogach.
  + sys: Do manipulacji ścieżką systemową i importami.
  + json: Do zarządzania danymi quizów i postępów w formacie JSON.
  + importlib: Do dynamicznego ładowania modułów quizów.
  + random: Do losowania pytań w quizach.
  + unittest: Do tworzenia i uruchamiania testów jednostkowych.
  + unittest.mock (w tym patch): Do mockowania wejścia/wyjścia w testach.
  + fractions.Fraction: (dla quizu z ułamkami) Wbudowana klasa Pythona do pracy z ułamkami.

**6. Sposób działania programu**

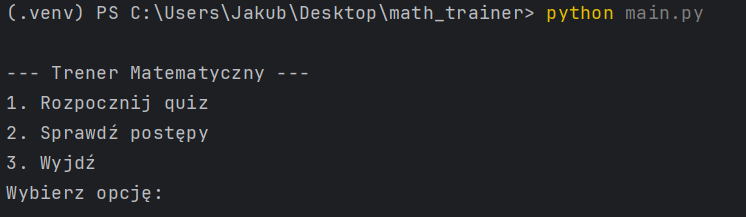
Najpierw w katalogu głównym projektu należy wpisać poniższą komendę, zainstaluje ona wszystkie potrzebne do działania programu zależności.

****

Następnie, w tym samym katalogu, możemy uruchomić program za pomocą komendy.

****

Przykładowe uruchomienie aplikacji:

****

**7. Przykłady kodu (z wyjaśnieniem)**

Fragment funkcji funkcyjnej:

Poniższe funkcje z modułu math\_trainer/core/utils.py są esencjonalnymi przykładami zastosowania **programowania funkcyjnego** w projekcie. Wykorzystują one wbudowane w Pythona funkcje wyższego rzędu (map, filter) w połączeniu z funkcjami anonimowymi (lambda), promując **deklaratywny styl programowania**, **niezmienność danych (immutability)** oraz **czystość funkcji**. Ich celem jest efektywna transformacja i agregacja danych bez modyfikowania stanu zewnętrznego aplikacji.

def filter\_positive\_numbers(numbers):

"""

Filtruje listę liczb, zwracając tylko liczby dodatnie.

Wykorzystuje funkcję filter() i lambda.

"""

return list(filter(lambda x: x > 0, numbers))

def square\_numbers(numbers):

"""

Przekształca listę liczb, zwracając listę ich kwadratów.

Wykorzystuje funkcję map() i lambda.

"""

return list(map(lambda x: x \* x, numbers))

def format\_progress\_data(progress\_records):

"""

Formatuje dane o postępach w bardziej czytelny sposób.

Wykorzystuje funkcję map() i lambda do tworzenia stringów.

Zakłada, że progress\_records to lista słowników np.

[{"quiz\_name": "Arytmetyka", "score": 8, "total": 10}]

"""

return list(map(lambda record: f"Quiz: {record['quiz\_name']}, Wynik: "

f"{record['score']}/{record['total']}", progress\_records))

def calculate\_average\_score(scores):

"""

Oblicza średnią z listy wyników.

Prosty przykład, który mógłby być częścią bardziej złożonej analizy.

"""

if not scores:

return 0.0

return sum(scores) / len(scores)

Fragment klasy:

import importlib

import os

import json

import random

import sys

class QuizManager:

"""Zarządza ładowaniem, uruchamianiem quizów i ich danymi z pliku JSON."""

*# Zmiana ścieżek dostosowana do nowej struktury pakietów*

def \_\_init\_\_(self, quiz\_dir="quizzes", quiz\_data\_file="../quizzes/quiz\_data.json"):

self.quiz\_dir = quiz\_dir

*# Ścieżka do quiz\_data.json względem katalogu 'core'*

self.quiz\_data\_file = os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), quiz\_data\_file))

self.available\_quizzes = self.\_load\_quiz\_definitions()

self.quiz\_questions = self.\_load\_quiz\_questions()

def \_load\_quiz\_questions(self):

"""Prywatna metoda do ładowania pytań quizowych z pliku JSON."""

questions\_data = {}

try:

with open(self.quiz\_data\_file, 'r', encoding='utf-8') as f:

questions\_data = json.load(f)

except FileNotFoundError:

print(f"Błąd: Plik z danymi quizów '{self.quiz\_data\_file}' nie znaleziono.")

except json.JSONDecodeError as e:

print(f"Błąd parsowania pliku JSON '{self.quiz\_data\_file}': {e}")

except Exception as e:

print(f"Wystąpił błąd podczas wczytywania danych quizów: {e}")

return questions\_data

def \_load\_quiz\_definitions(self):

"""

Prywatna metoda do ładowania klas quizów z katalogu quizzes.

"""

quizzes = {}

*# Ścieżka do katalogu quizzes względem katalogu core*

quiz\_module\_path = os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), '..', self.quiz\_dir))

*# Dodaj katalog quizzes do ścieżki sys.path TYLKO na czas ładowania definicji*

*# To jest kluczowe, aby importlib.import\_module mógł znaleźć pliki .py bezpośrednio*

*# bez konieczności odwoływania się do 'math\_trainer.quizzes' jako pełnego pakietu.*

*# Jest to obejście dla dynamicznego importowania, gdy struktura pakietów jest złożona.*

if quiz\_module\_path not in sys.path:

sys.path.insert(0, quiz\_module\_path)

for filename in os.listdir(quiz\_module\_path):

if filename.endswith(".py") and filename != "\_\_init\_\_.py":

module\_name = filename[:-3]

try:

module = importlib.import\_module(module\_name)

for attr\_name in dir(module):

attribute = getattr(module, attr\_name)

if isinstance(attribute, type) and 'Quiz' in attr\_name:

if hasattr(attribute, 'get\_name') and callable(getattr(attribute, 'get\_name')):

quiz\_instance = attribute()

quizzes[quiz\_instance.get\_name()] = attribute

else:

print(

f"Ostrzeżenie: Klasa {attr\_name} w module {module\_name} nie ma metody 'get\_name'. Zostaje pominięta.")

except Exception as e:

print(f"Błąd ładowania definicji quizu z pliku {filename}: {e}")

if quiz\_module\_path in sys.path:

sys.path.remove(quiz\_module\_path)

importlib.invalidate\_caches()

return quizzes

def list\_quizzes(self):

"""Zwraca listę dostępnych nazw quizów na podstawie danych JSON."""

return list(self.quiz\_questions.keys())

def get\_quiz\_instance\_and\_questions(self, quiz\_name, num\_questions=5):

"""

Zwraca instancję wybranej klasy quizu i zestaw pytań dla niego.

Pytania są pobierane z pliku JSON.

"""

quiz\_class = self.available\_quizzes.get(quiz\_name)

if not quiz\_class:

raise ValueError(f"Definicja quizu '{quiz\_name}' nie znaleziono w modułach Python.")

quiz\_instance = quiz\_class()

questions\_for\_quiz = self.quiz\_questions.get(quiz\_name)

if not questions\_for\_quiz:

raise ValueError(f"Brak pytań dla quizu '{quiz\_name}' w pliku quiz\_data.json.")

if not isinstance(questions\_for\_quiz, list):

raise TypeError(f"Oczekiwano listy pytań dla quizu '{quiz\_name}', otrzymano {type(questions\_for\_quiz)}")

*# Wybierz losowe pytania, jeśli jest ich więcej niż num\_questions*

if len(questions\_for\_quiz) > num\_questions:

selected\_questions = random.sample(questions\_for\_quiz, num\_questions)

else:

selected\_questions = questions\_for\_quiz

for q in selected\_questions:

if not isinstance(q, dict) or 'question' not in q or 'answer' not in q:

raise ValueError(f"Nieprawidłowy format pytania w quiz\_data.json dla quizu '{quiz\_name}': {q}")

return quiz\_instance, selected\_questions

def run\_quiz(self, quiz\_instance, questions):

"""Przeprowadza quiz i zwraca liczbę poprawnych odpowiedzi."""

if not hasattr(quiz\_instance, 'check\_answer') or not callable(getattr(quiz\_instance, 'check\_answer')):

raise TypeError("Obiekt quizu musi mieć metodę 'check\_answer(user\_answer\_str, correct\_answer)'.")

score = 0

total\_questions = len(questions)

print(f"\n--- Rozpoczynam quiz: {quiz\_instance.get\_name()} ---")

for i, q\_data in enumerate(questions):

question\_str = q\_data['question']

correct\_answer = q\_data['answer']

print(f"\nPytanie {i + 1}/{total\_questions}: {question\_str}")

try:

user\_answer\_str = input("Twoja odpowiedź: ").strip()

is\_correct = quiz\_instance.check\_answer(user\_answer\_str, correct\_answer)

if is\_correct:

print("Poprawna odpowiedź!")

score += 1

else:

print(f"Błędna odpowiedź. Prawidłowa odpowiedź to: {correct\_answer}")

except ValueError as e:

print(f"Błąd wejścia: {e}. Spróbuj ponownie.")

except ZeroDivisionError:

print("Wystąpił błąd dzielenia przez zero. To pytanie zostało pominięte.")

except Exception as e:

print(f"Wystąpił nieoczekiwany błąd podczas quizu: {e}")

print(f"\n--- Koniec quizu: {quiz\_instance.get\_name()} ---")

print(f"Twój wynik: {score}/{total\_questions}")

return score, total\_questions

Klasa QuizManager jest rdzeniem zarządzającym quizami w aplikacji, demonstrując kilka kluczowych zasad OOP:

1. **Enkapsulacja (Encapsulation):**
   * QuizManager **hermetyzuje** cały stan i logikę związaną z quizami. Przechowuje informacje o **dostępnych definicjach klas quizów** (self.available\_quizzes) oraz **załadowanych pytaniach** (self.quiz\_questions). Te wewnętrzne dane są dostępne tylko za pośrednictwem publicznych metod klasy (np. list\_quizzes(), get\_quiz\_instance\_and\_questions()), co chroni je przed niekontrolowanym dostępem i modyfikacjami z zewnątrz.
   * Prywatne metody, takie jak \_load\_quiz\_definitions() i \_load\_quiz\_questions(), są prefiksowane podkreślnikiem (\_), sygnalizując, że są to wewnętrzne mechanizmy klasy, które powinny być wywoływane tylko przez samą klasę QuizManager.
2. **Abstrakcja (Abstraction):**
   * QuizManager **abstrakcjonuje** złożoność ładowania i zarządzania różnymi typami quizów. Użytkownik kodu (np. moduł main.py) nie musi wiedzieć, *jak* QuizManager importuje moduły quizów, *jak* wczytuje pytania z JSON-a, czy *jak* sprawdza konkretne odpowiedzi. Wystarczy, że wywoła metody takie jak list\_quizzes() czy get\_quiz\_instance\_and\_questions(), a QuizManager zajmie się szczegółami.
   * Interfejs, który muszą implementować poszczególne klasy quizów (metody get\_name() i check\_answer()), jest formą abstrakcji, która pozwala QuizManagerowi na jednolite traktowanie wszystkich rodzajów quizów, niezależnie od ich wewnętrznej logiki.
3. **Delegacja Odpowiedzialności (Delegation):**
   * Zamiast QuizManager samodzielnie sprawdzać odpowiedzi na pytania, **deleguje** to zadanie do konkretnej **instancji quizu** (quiz\_instance.check\_answer(user\_answer\_str, correct\_answer) w metodzie run\_quiz). QuizManager odpowiada za ogólny przepływ quizu (zadawanie pytań, zliczanie punktów), ale to specyficzny obiekt quizu (np. BasicArithmeticQuiz czy FractionQuiz) wie, jak zweryfikować odpowiedź adekwatnie do swojego typu. To jest zgodne z zasadą **Pojedynczej Odpowiedzialności (Single Responsibility Principle)**.
4. **Polimorfizm (Polymorphism):**
   * Metoda run\_quiz() demonstruje polimorfizm. Przyjmuje ona argument quiz\_instance, który może być instancją dowolnej klasy quizu (np. BasicArithmeticQuiz, FractionQuiz, PowersQuiz), pod warunkiem, że wszystkie te klasy implementują metodę check\_answer(). QuizManager wywołuje quiz\_instance.check\_answer(), a Python automatycznie decyduje, która konkretna implementacja tej metody zostanie użyta, w zależności od rzeczywistego typu obiektu quiz\_instance. Pozwala to na elastyczne i rozszerzalne dodawanie nowych rodzajów quizów w przyszłości.
5. **Dynamiczne Ładowanie Klas (Reflection/Metaprogramming):**
   * Metoda \_load\_quiz\_definitions() wykorzystuje importlib do **dynamicznego importowania modułów** na podstawie plików w katalogu quizzes. Następnie introspekcja (dir(), getattr(), isinstance()) jest używana do znalezienia i załadowania odpowiednich klas quizów. Jest to zaawansowana technika OOP, która umożliwia dodawanie nowych klas quizów do systemu bez modyfikowania kodu QuizManager – wystarczy umieścić nowy plik .py w odpowiednim katalogu, a QuizManager sam go wykryje i załaduje.

Podsumowując, klasa QuizManager jest filarem architektury obiektu w projekcie. Dzięki zastosowaniu enkapsulacji, abstrakcji, delegacji, polimorfizmu i dynamicznego ładowania, zapewnia ona solidną, rozszerzalną i łatwą w utrzymaniu platformę do zarządzania różnorodnymi quizami matematycznymi.

Obsługa wyjątków:

def get\_positive\_integer\_input(prompt):

"""Pobiera od użytkownika dodatnią liczbę całkowitą."""

while True:

try:

value\_str = input(prompt)

value = int(value\_str)

if value <= 0:

print("Wprowadź liczbę większą od zera.")

else:

return value

except ValueError:

print("Nieprawidłowy format. Wprowadź liczbę całkowitą.")

1. **Blok try:** Instrukcje zawarte w bloku try są wykonywane jako pierwsze. W tym przypadku jest to próba pobrania danych od użytkownika za pomocą input() i natychmiastowa próba konwersji tego tekstu na liczbę całkowitą (int()). To właśnie ta konwersja jest potencjalnym źródłem błędu, jeśli użytkownik wprowadzi niepoprawne dane.
2. **Walidacja Biznesowa:** Po udanej konwersji, kod sprawdza, czy wprowadzona wartość spełnia dodatkowe kryterium biznesowe – czy jest większa od zera (if value > 0). Jeśli nie, użytkownik otrzymuje odpowiedni komunikat, a pętla while True zapewnia, że proces wprowadzania danych zostanie powtórzony.
3. **Blok except ValueError:** Jest to specyficzny blok obsługujący wyjątek ValueError. Ten typ wyjątku jest zgłaszany przez Pythona, gdy operacja (w tym przypadku int()) otrzymuje argument o odpowiednim typie, ale nieprawidłowej wartości – np. próba konwersji ciągu znaków "abc" na liczbę całkowitą. Gdy ValueError zostanie przechwycony, program nie przerywa działania (nie "crashuje"), lecz wykonuje kod z bloku except, informując użytkownika o błędzie i prosi o ponowne wprowadzenie danych. Jest to kluczowe dla **użyteczności i stabilności** aplikacji.
4. **Blok except Exception as e:** Ten ogólny blok except przechwytuje wszelkie inne, nieprzewidziane wyjątki, które mogłyby wystąpić podczas wykonywania kodu w bloku try. Służy jako "catch-all", zapewniając, że aplikacja nie zakończy działania w niekontrolowany sposób. Pozwala to na zdiagnozowanie problemu i ewentualne logowanie błędów, jednocześnie informując użytkownika o problemie i pozwalając mu na ponowną próbę.
5. **Pętla while True:** Użycie nieskończonej pętli while True w połączeniu z instrukcją return value (która przerywa pętlę i zwraca wartość) jest wzorcem projektowym w obsłudze wejścia użytkownika. Zapewnia to, że funkcja będzie **nieustannie prosić o wprowadzenie danych**, dopóki nie otrzyma wartości, która jest zarówno prawidłową liczbą całkowitą, jak i spełnia kryterium bycia dodatnią. To gwarantuje, że funkcja zawsze zwróci poprawny wynik lub będzie czekać na prawidłowe dane wejściowe.

Podsumowując, zastosowanie bloków try-except w funkcji get\_positive\_integer\_input jest fundamentalnym elementem. Skutecznie zapobiega awariom programu spowodowanym błędnym działaniem użytkownika i zapewnia płynne, stabilne doświadczenie, co jest kluczowe dla interaktywnych aplikacji konsolowych. Podobne mechanizmy są zastosowane w innych częściach kodu (np. przy ładowaniu plików JSON czy sprawdzaniu odpowiedzi quizów), co tworzy spójny system obsługi błędów w całym projekcie.

**8. Testowanie**

Zapewnienie wysokiej jakości i niezawodności aplikacji jest ważnym elementem procesu jej tworzenia. W tym celu w projekcie zastosowano kompleksowe podejście do testowania oprogramowania, koncentrujące się na **testach jednostkowych**, które są nieodzowne do weryfikacji poprawności działania poszczególnych komponentów systemu w izolacji.

Głównym narzędziem wykorzystanym do implementacji testów był wbudowany w Pythona moduł **unittest**. Jest to standardowa biblioteka, która dostarcza struktury i narzędzi niezbędnych do pisania zautomatyzowanych testów. Wszystkie testy zostały logicznie zorganizowane w osobnym pakiecie tests/, który odzwierciedla strukturę katalogów kodu źródłowego. Taka organizacja znacząco ułatwia zarządzanie testami, ich lokalizację oraz uruchamianie.

Każda klasa testowa w projekcie dziedziczy po unittest.TestCase, co natychmiastowo udostępnia bogaty zestaw metod asercji, takich jak assertEqual() do sprawdzania równości, assertTrue()/assertFalse() do weryfikacji warunków logicznych, czy assertRaises() do potwierdzania, że dany kod poprawnie zgłasza oczekiwany wyjątek. Ponadto, metoda setUp() jest konsekwentnie wykorzystywana do inicjalizacji wspólnych zasobów (np. tworzenia instancji quizu) przed wykonaniem każdego testu, co gwarantuje, że każde środowisko testowe jest czyste i niezależne.

**Korzyści wynikające z przyjęcia takiego podejścia do testowania:**

Przede wszystkim, umożliwia ono automatyzację procesu weryfikacji funkcjonalności, co jest kluczowe w dynamicznym środowisku programistycznym. Automatyczne uruchamianie testów pozwala szybko wykrywać błędy i regresje. Po drugie, izolacja testów jednostkowych ułatwia precyzyjne lokalizowanie źródła problemu – jeśli test komponentu A zawodzi, wiadomo, że problem leży prawdopodobnie w tym komponencie. Po trzecie, testy służą jako żywa dokumentacja oczekiwanego zachowania kodu – czytając testy, można zrozumieć, co dany fragment kodu powinien robić w różnych scenariuszach. Wreszcie, regularne uruchamianie testów regresyjnych zapewnia, że nowe zmiany lub dodawane funkcjonalności nie wprowadzają błędów do już działających, wcześniej przetestowanych części aplikacji.

Integralnym elementem skutecznego testowania jest szczególne uwzględnienie przypadków granicznych (edge cases). Są to wartości lub scenariusze na krańcach dopuszczalnego zakresu, lub te o nietypowych charakterystykach, które statystycznie częściej prowadzą do ujawnienia błędów w logice programu.

**9. Wnioski**

Projekt "Trener Matematyczny" to prosta aplikacja, która pomaga ćwiczyć matematykę. Zbudowaliśmy ją tak, żeby działała stabilnie i była łatwa do rozbudowy.

**Co udało się osiągnąć:**

1. **Dobrze Zbudowany Kod:** Użyliśmy dwóch ważnych sposobów pisania kodu (Programowania Obiektowego i Funkcyjnego). Dzięki temu program jest uporządkowany, a każda jego część ma swoje zadanie. To sprawia, że kod jest łatwiejszy do zrozumienia i zmiany.
2. **Odporność na Błędy:** Aplikacja jest gotowa na to, że użytkownik może wpisać coś źle. Dzięki temu program się nie zawiesza, tylko prosi o poprawne dane. To sprawia, że korzystanie z niego jest przyjemniejsze.
3. **Dobre Testowanie:** Sporo czasu poświęciliśmy na sprawdzanie, czy program działa poprawnie. Główne testy (testy jednostkowe) dokładnie sprawdzają małe części kodu, a także radzenie sobie z nietypowymi sytuacjami (przypadki graniczne). Dzięki temu mamy pewność, że program jest niezawodny.

**Co można poprawić w przyszłości:**

* **Współpraca Modułów:** Możemy dodać testy, które sprawdzą, czy większe części programu (moduły) dobrze ze sobą współpracują.
* **Szybkość i Pamięć:** W przyszłości można zbadać, czy program działa szybko i nie zużywa za dużo pamięci, zwłaszcza przy większej ilości danych.
* **Nowe Funkcje:** Łatwo będzie dodawać nowe rodzaje quizów lub zmienić wygląd aplikacji na bardziej graficzny (zamiast tekstowego).

Podsumowując, "Trener Matematyczny" to solidny program, który dobrze sprawdza się w swoim zadaniu. Jego budowa pozwala na łatwe dodawanie nowych funkcji i ciągłe ulepszanie.