





N r	Obszar	Wymaganie	KOD	JE ST	Przyz nane pkt	Pk t m ax
		Wprowadzanie danych	<pre># quiz_project/quiz_creator/creator.py (fragmenty z metody QuizCreator.create_new_quiz)  # Wprowadzanie tytułu quizu title = input("Podaj tytuł quizu (np. 'Geografia Polski'): ").strip()  # Wprowadzanie opisu quizu description = input("Podaj krótki opis quizu (opcjonalnie): ").strip()  # Wprowadzanie treści pytania question_text = input("Wpisz treść pytania (naciśnij Enter, aby zakończyć dodawanie pytań): ").strip()  # Wprowadzanie opcji odpowiedzi (w pętli) option = input(f"Opcja {option_count}: ").strip()  # Wprowadzanie numeru poprawnej odpowiedzi user_input = input("Wpisz numer poprawnej odpowiedzi: ").strip()</pre>	<input checked="" type="checkbox"/>	2	2
		Wyświetlanie danych	<pre># quiz_project/quiz_player/player.py (fragmenty z metod QuizPlayer.play_quiz)  # Wyświetlanie dostępnych quizów print("Dostępne quizy:") for i, quiz_name in enumerate(available_quizzes):     print(f" {i + 1}. {quiz_name}")  # Wyświetlanie tytułu i opisu quizu print(f"\n--- Rozpoczęcie quizu: {quiz.title} ---") if quiz.description:     print(f"Opis: {quiz.description}")  # Wyświetlanie pytania i opcji (metoda Question.display() wewnętrznie używa print)</pre>	<input checked="" type="checkbox"/>	2	2


	<pre> print(f"\n--- Pytanie {i + 1}/{total_questions} ---") print(question.display()) # Gdzie question.display() to: # Pytanie: {self.question_text}\n 1. {opcja1}\n 2. {opcja2}\n...  # Wyświetlanie wyników końcowych print("\n--- Koniec quizu! ---") print(f"Twój wynik: {correct_answers_count}/{total_questions} poprawnych odpowiedzi.") </pre>			
Zmiana danych	<pre> # quiz_project/quiz_creator/creator.py (fragment z metody QuizCreator.edit_existing_quiz)  # ... (kod wyboru quizu do edycji) ...  # Główna pętla edycji while True:     # ... (wyświetlanie menu edycji) ...     edit_choice = input("Wybierz opcję edycji (1- 6): ").strip()      if edit_choice == '1': # Edycja tytułu i opisu         new_title = input(f"Nowy tytuł quizu (obecny: '{quiz_to_edit.title}'): ").strip()         if new_title:             quiz_to_edit.title = new_title         # ... (logika dla opisu) ...      elif edit_choice == '3': # Edycja istniejącego pytania         # ... (wyświetlanie pytań i wybór pytania do edycji) ...         question_to_edit = quiz_to_edit.questions[q_index]          # Edycja treści pytania         new_q_text = input(f"Nowa treść pytania (obecna: '{question_to_edit.question_text}'): ").strip()         if new_q_text:             question_to_edit.question_text = new_q_text </pre>		2	2


	<pre> # Edycja opcji odpowiedzi new_options = [] for i, opt in enumerate(question_to_edit.options):     edited_opt = input(f"Opcja {i+1} (obecna: '{opt}'): ").strip()     new_options.append(edited_opt if edited_opt else opt) # ... (logika dodawania nowych opcji) ...  # Edycja poprawnej odpowiedzi new_correct_input = input(f"Nowy numer poprawnej odpowiedzi (obecny: {current_correct + 1}): ").strip() # ... (konwersja i walidacja) ... # ... (inne opcje, zapis) ... </pre>			
Wyszukiwa nie danych	<pre> # quiz_project/quiz_data/manager.py (fragment metody QuizDataManager.list_available_quizzes)  class QuizDataManager:     # ... (inne metody) ...      @staticmethod     def list_available_quizzes(directory: str = "data/quiz_examples") -&gt; list[str]:         """         Lists all available quiz files (JSON files) in the specified directory.          Args:             directory (str): The directory to search for quiz files.              Defaults to "data/quiz_examples".          Returns:             list[str]: A list of filenames (without the .json extension) representing available quizzes. Returns an empty list </pre>		2	2

		<pre>         if the directory does not exist or contains         no quizzes.         """          if not os.path.exists(directory):             return [] # Return empty list if directory doesn't             exist          quiz_files = []         try:             for item in os.listdir(directory):                 if item.endswith(".json"):                     quiz_files.append(os.path.splitext(item)[0]) #                     Add filename without extension             quiz_files.sort() # Sort alphabetically for             consistent display         except OSError as e:             print(f"Error listing files in directory {directory}:             {e}")         return quiz_files  # quiz_project/quiz_player/player.py (fragment metody QuizPlayer.play_quiz)  class QuizPlayer:     # ... (inne metody) ...      @staticmethod     def play_quiz():         """         Allows the user to select and play an existing quiz.         """         print("\n--- Rozpoczęcie odtwarzania quizu ---")          # Wyszukiwanie (listowanie) dostępnych quizów         available_quizzes =         QuizDataManager.list_available_quizzes()          if not available_quizzes:             print("Brak dostępnych quizów. Najpierw utwórz             quiz w trybie kreatora.")             return </pre>		
--	--	---	--	--


		<pre> print("Dostępne quizy:") for i, quiz_name in enumerate(available_quizzes):     print(f" {i + 1}. {quiz_name}")  # ... (kod wyboru quizu przez użytkownika) ... </pre>			
Przedstawi enie wyników		<pre> # quiz_project/quiz_player/player.py (fragmenty z metody QuizPlayer.play_quiz)  class QuizPlayer:     # ... (inne metody i kod) ...      @staticmethod     def play_quiz():         # ... (logika odtwarzania pytań i zbierania odpowiedzi) ...          print("\n--- Koniec quizu! ---")         # Prezentacja ogólnego wyniku tekstowo         print(f"Twój wynik: {correct_answers_count}/{total_questions} poprawnych odpowiedzi.")          # ... (analiza wyników z map/filter) ...          # Wizualizacja i zapis wykresu wyników  QuizPlayer.generate_and_save_results_chart(num_correct , num_incorrect, quiz.title)          print("\nSzczegółowe wyniki zostały zapisane w raporcie graficznym.")      @staticmethod     def generate_and_save_results_chart(correct_count: int, incorrect_count: int, quiz_title: str):         """         Generates a pie chart showing the distribution of correct vs. incorrect answers and saves it to the 'reports' directory.         """ </pre>		2	2

			<pre># ... (kod generujący wykres matplotlib) ...  # Komunikat o zapisaniu wykresu chart_filename = f"wyniki_{quiz_title.replace(' ', ' ').lower()}.png" chart_filepath = os.path.join("reports", chart_filename) print(f"Wykres wyników został zapisany w: {chart_filepath}")</pre>			
2	Podstawy	Zmienne	<pre># models/question.py (fragment)  class Question:     def __init__(self, question_text: str, options: list, correct_answer_index: int):         self.question_text = question_text.strip() # Zmienna instancji         self.options = [opt.strip() for opt in options] # Zmienna instancji (lista)         self.correct_answer_index = correct_answer_index # Zmienna instancji  # quiz_creator/creator.py (fragment z metody create_new_quiz)  class QuizCreator:     @staticmethod     def create_new_quiz():         title = input("Podaj tytuł quizu...").strip() # Zmienna lokalna         description = input("Podaj krótki opis...").strip() # Zmienna lokalna         new_quiz = Quiz(title, description) # Zmienna lokalna, przechowująca obiekt  # quiz_player/player.py (fragment z metody play_quiz)  class QuizPlayer:     @staticmethod     def play_quiz():</pre>		2	2


		<pre>correct_answers_count = 0 # Zmienna do zliczania poprawnych odpowiedzi total_questions = len(quiz.questions) # Zmienna przechowująca liczbę całkowitą user_answers = [] # Zmienna lokalna, przechowująca listę słowników</pre>			
	typy danych	<pre># models/question.py (fragmenty)  class Question:     def __init__(self, question_text: str, options: list, correct_answer_index: int):         # Typ danych: str (ciąg znaków)         self.question_text = question_text.strip()          # Typ danych: list (lista stringów)         self.options = [opt.strip() for opt in options]          # Typ danych: int (liczba całkowita)         self.correct_answer_index = correct_answer_index  # quiz_data/manager.py (fragment)  class QuizDataManager:     @staticmethod     def save_quiz(quiz: Quiz, filename: str, directory: str = "data/quiz_examples"):         # Typ danych: bool (wartość logiczna)         if not isinstance(quiz, Quiz):             raise TypeError("Only Quiz objects can be saved.")  # quiz_player/player.py (fragment)  class QuizPlayer:     @staticmethod     def play_quiz():         correct_answers_count = 0 # Typ danych: int (liczba całkowita)         user_answers = [] # Typ danych: list (lista słowników)</pre>		2	2


	<pre># Typ danych: dict (słownik) user_answers.append({     "question_text": "Przykład pytania",     "is_correct": True # Typ danych: bool (wartość logiczna) })</pre>			
komentarze	<pre># models/question.py (fragmenty)  class Question:     """     Represents a single question in a quiz. # Docstring dla klasy     """      def __init__(self, question_text: str, options: list, correct_answer_index: int):         # Sprawdzanie, czy tekst pytania nie jest pusty         if not isinstance(question_text, str) or not question_text.strip():             raise ValueError("Question text cannot be empty.")          self.question_text = question_text.strip() # Usunięcie białych znaków na początku i końcu  # quiz_creator/creator.py (fragment)  class QuizCreator:     @staticmethod     def create_new_quiz():         """         Guides the user through creating a new quiz, question by question. # Docstring dla metody         Collects quiz title, description, and then iteratively collects questions         until the user decides to stop. Finally, it saves the quiz.         """          # Get quiz title         while True: # Pętla do wymuszenia niepustego tytułu</pre>		1	1





	<pre> title = input("Podaj tytuł quizu...").strip()  # quiz_data/manager.py (fragment)  class QuizDataManager:     @staticmethod     def save_quiz(quiz: Quiz, filename: str, directory: str = "data/quiz_examples"):         """         Saves a Quiz object to a JSON file. # Docstring dla metody         """         if not filename.endswith(".json"):             filename += ".json" # Zapewnienie rozszerzenia .json          # Ensure the directory exists         if not os.path.exists(directory): # Sprawdzenie istnienia katalogu             os.makedirs(directory) # Utworzenie katalogu </pre>			
operatory	<pre> # models/question.py (fragmenty)  class Question:     def __init__(self, question_text: str, options: list, correct_answer_index: int):         # Operator przypisania (=) i operator wywołania metody (.)         self.question_text = question_text.strip()          # Operator porównania (!=) i operator logiczny (not)         if not isinstance(question_text, str) or not question_text.strip():             raise ValueError("Question text cannot be empty.")          # Operator porównania (&lt;=, &lt;) i operator logiczny (and)         if not isinstance(correct_answer_index, int) or not (0 &lt;= correct_answer_index &lt; len(options)): </pre>		1.5	1, 5

	<pre>         raise ValueError("Correct answer index is out of         bounds or not an integer.")      def is_correct(self, user_answer_index: int) -&gt; bool:         # Operator porównania (==)         return user_answer_index ==         self.correct_answer_index  # quiz_player/player.py (fragmenty z metody QuizPlayer.play_quiz)  class QuizPlayer:     @staticmethod     def play_quiz():         correct_answers_count = 0 # Operator przypisania         (=)         total_questions = len(quiz.questions) # Operator         przypisania (=)          # Operator arytmetyczny (+) do inkrementacji         correct_answers_count += 1          # Operator arytmetyczny (-) do konwersji numeru         opcji na indeks         answer_index = int(user_input) - 1          # Operator logiczny (not) i operator porównania         (==)         # correct_results = list(filter(lambda ans:         ans["is_correct"], user_answers))         incorrect_results = list(filter(lambda ans: not         ans["is_correct"], user_answers))          # Operator formatowania f-string (f"...")         print(f"Twój wynik:         {correct_answers_count}/{total_questions} poprawnych         odpowiedzi.")  # quiz_creator/creator.py (fragment z metody QuizCreator._save_quiz_with_prompt) </pre>			
--	--	--	--	--


	<pre> @staticmethod def _save_quiz_with_prompt(quiz: Quiz):     # Operator przypisania (=) i operator wywołania     metody (.)     filename = input("Podaj nazwę pliku...").strip()      # Operator logiczny (or) i operator przynależności     (in)     if not all(c.isalnum() or c in ['- ', '_'] for c in filename):         print("Nazwa pliku może zawierać tylko litery,         cyfry, myślniki i podkreślenia.") </pre>			
Instrukcje warunkowe (if, elif, else)	<pre> # quiz_project/main.py (fragment z funkcji main)  def main():     # ...     while True:         display_menu()         choice = input("Wybierz opcję (1-4): ").strip()          if choice == '1': # Warunek: jeśli wybór to '1'             (utwórz quiz)             QuizCreator.create_new_quiz()         elif choice == '2': # Warunek: jeśli wybór to '2'             (odtworzyć quiz)             QuizPlayer.play_quiz()         elif choice == '3': # Warunek: jeśli wybór to '3'             (edytuj quiz)             QuizCreator.edit_existing_quiz()         elif choice == '4': # Warunek: jeśli wybór to '4'             (wyjdź)             print("Dziękujemy za skorzystanie z aplikacji. Do             widzenia!")             break         else: # Warunek: jeśli żaden z powyższych nie jest             spełniony             print("Nieprawidłowy wybór. Proszę wybrać             opcję 1, 2, 3 lub 4.")     # ...  # quiz_project/models/question.py (fragment z konstruktora Question) </pre>		3	3

	<pre> class Question:     def __init__(self, question_text: str, options: list, correct_answer_index: int):         if not isinstance(question_text, str) or not question_text.strip(): # Warunek: walidacja tekstu pytania             raise ValueError("Question text cannot be empty.")         if not isinstance(options, list) or not options: # Warunek: walidacja listy opcji             raise ValueError("Options must be a non-empty list.")         # ... (inne warunki) ...  # quiz_project/quiz_player/player.py (fragment z metody play_quiz)  class QuizPlayer:     @staticmethod     def play_quiz():         # ...         if not available_quizzes: # Warunek: jeśli brak dostępnych quizów             print("Brak dostępnych quizów. Najpierw utwórz quiz w trybie kreatora.")             return         # ...         if quiz.description: # Warunek: jeśli quiz ma opis             print(f"Opis: {quiz.description}")         # ...         if question.is_correct(answer_index): # Warunek: jeśli odpowiedź poprawna             print("Poprawna odpowiedź!")         else: # Warunek: jeśli odpowiedź niepoprawna             print(f"Niepoprawna odpowiedź. Poprawna to: {question.options[question.correct_answer_index]}") </pre>			
Instrukcje iteracyjne				
for	# quiz_project/quiz_player/player.py (fragment z metody QuizPlayer.play_quiz)		2	2

	<pre> class QuizPlayer:     @staticmethod     def play_quiz():         # ... (kod wcześniejszy) ...          # Pętla 'for' do iteracji przez listę dostępnych         quizów         print("Dostępne quizy:")         for i, quiz_name in enumerate(available_quizzes):             print(f" {i + 1}. {quiz_name}")          # Pętla 'for' do iteracji przez pytania w quizie         for i, question in enumerate(quiz.questions):             print(f"\n--- Pytanie {i + 1}/{total_questions} ---")             print(question.display())             # ... (logika odpowiedzi) ...          # Pętla 'for' do wyświetlania pytań, na które         odpowiedzią błędnie         if incorrect_question_texts:             print("\nPytań, na które odpowiedział/aś             błędnie:")             for text in incorrect_question_texts:                 print(f"- {text}") </pre>			
while	<pre> # quiz_project/quiz_creator/creator.py (fragment z metody QuizCreator.create_new_quiz)  class QuizCreator:     @staticmethod     def create_new_quiz():         # Pętla 'while' do wymuszenia wprowadzenia         nie pustego tytułu         while True:             title = input("Podaj tytuł quizu (np. 'Geografia Polski'): ").strip()             if title:                 break # Wyjście z pętli, gdy tytuł jest                 prawidłowy             else: </pre>		2	2

	<pre> print("Tytuł quizu nie może być pusty. Spróbuj ponownie.")  # Pętla 'while' do zbierania opcji odpowiedzi, dopóki użytkownik nie zakończy options = [] option_count = 1 print("Wpisuj opcje odpowiedzi. Naciśnij Enter na pustej linii, aby zakończyć dodawanie opcji.") while True:     option = input(f"Opcja {option_count}: ").strip()     if not option:         if len(options) &lt; 2:             print("Pytanie musi mieć co najmniej dwie opcje odpowiedzi.")             continue # Kontynuuj pętlę, dopóki nie będzie wystarczającej liczby opcji         else:             break # Wyjście z pętli, gdy opcje są prawidłowe     # ... (logika dodawania opcji) ...  # quiz_project/main.py (fragment z funkcji main)  def main():     # Główna pętla programu, działająca dopóki użytkownik nie wybierze opcji wyjścia     while True:         display_menu()         choice = input("Wybierz opcję (1-4): ").strip()         # ... (obsługa wyboru) ...         if choice == '4':             break # Wyjście z pętli głównej </pre>			
Operacje wejścia (input)	<pre> # quiz_project/main.py (fragment z funkcji main)  def main():     # ...     while True:         display_menu()         # Zbieranie wyboru użytkownika z menu         choice = input("Wybierz opcję (1-4): ").strip() </pre>		1.5	1, 5


	<pre> # ...  # quiz_project/quiz_creator/creator.py (fragmenty z metody QuizCreator.create_new_quiz)  class QuizCreator:     @staticmethod     def create_new_quiz():         # Zbieranie tytułu quizu         title = input("Podaj tytuł quizu (np. 'Geografia Polski'): ").strip()         # Zbieranie opisu quizu         description = input("Podaj krótki opis quizu (opcjonalnie): ").strip()          # Zbieranie treści pytania         question_text = input("Wpisz treść pytania (naciśnij Enter, aby zakończyć dodawanie pytań): ").strip()          # Zbieranie opcji odpowiedzi w pętli         option = input(f"Opcja {option_count}: ").strip()          # Zbieranie numeru poprawnej odpowiedzi         user_input = input("Wpisz numer poprawnej odpowiedzi: ").strip()  # quiz_project/quiz_player/player.py (fragmenty z metody QuizPlayer.play_quiz)  class QuizPlayer:     @staticmethod     def play_quiz():         # Zbieranie wyboru quizu do odtworzenia         choice = input("Wybierz numer quizu do odtworzenia: ").strip()          # Zbieranie odpowiedzi użytkownika na pytanie         user_input = input("Wpisz numer odpowiedzi: ").strip() </pre>			
--	--	--	--	--


	<pre># quiz_project/quiz_creator/creator.py (fragment z metody QuizCreator.edit_existing_quiz)  @staticmethod def edit_existing_quiz():     # ...     # Zbieranie wyboru quizu do edycji     choice = input("Wybierz numer quizu do edycji: ").strip()     # ...     # Zbieranie wyboru opcji edycji     edit_choice = input("Wybierz opcję edycji (1-6): ").strip()     # ...     # Zbieranie nowego tytułu i opisu     new_title = input(f"Nowy tytuł quizu (obecny: '{quiz_to_edit.title}'): ").strip()     new_description = input(f"Nowy opis quizu (obecny: '{quiz_to_edit.description}'): ").strip()     # ...     # Zbieranie numeru pytania do edycji/usunięcia     q_index_input = input("Wpisz numer pytania do edycji: ").strip()     # ...     # Zbieranie edytowanej treści pytania i opcji     new_q_text = input(f"Nowa treść pytania (obecna: '{question_to_edit.question_text}'): ").strip()     edited_opt = input(f"Opcja {i+1} (obecna: '{opt}'): ").strip()     # ...     # Zbieranie potwierdzenia nadpisania pliku     overwrite = input(f"Plik '{{filename}}.json' już istnieje. Czy chcesz go nadpisać? (tak/nie): ").lower()</pre>			
Operacje wyjścia (print)	<pre># quiz_project/main.py (fragment z funkcji main)  def main():     print("Witaj w Aplikacji Quizowej!") # Powitalna wiadomość     while True:         # Wyświetlanie menu głównego         print("\n--- Menu Główne ---")</pre>		1.5	1, 5




	<pre> print("1. Utwórz nowy quiz") print("2. Odtwórz quiz") print("3. Edytuj istniejący quiz") print("4. Wyjdź") print("-----") # ... if choice == '4':     print("Dziękujemy za skorzystanie z aplikacji. Do widzenia!") # Wiadomość pożegnalna else:     print("Nieprawidłowy wybór. Proszę wybrać opcję 1, 2, 3 lub 4.") # Komunikat o błędzie  # quiz_project/quiz_creator/creator.py (fragmenty z metod QuizCreator.create_new_quiz i _add_questions_to_quiz)  class QuizCreator:     @staticmethod     def create_new_quiz():         print("\n--- Rozpoczęcie tworzenia nowego quizu - ---") # Informacja o trybie         # ...         print(f"Quiz '{title}' został utworzony. Teraz dodaj pytania.") # Potwierdzenie utworzenia quizu         # ...         if not new_quiz.questions:             print("Nie dodano żadnych pytań. Quiz nie zostanie zapisany.") # Komunikat o niezapisaniu pustego quizu             return         # ...         print("Quiz został pomyślnie zapisany!") # Potwierdzenie zapisu      @staticmethod     def _add_questions_to_quiz(quiz: Quiz):         print("\n--- Dodawanie nowego pytania ---") # Nagłówek sekcji dodawania pytania         # ... </pre>			
--	--	--	--	--




	<pre> print("Pytanie dodane pomyślnie!") # Potwierdzenie dodania pytania  # quiz_project/quiz_player/player.py (fragmenty z metod QuizPlayer.play_quiz i generate_and_save_results_chart)  class QuizPlayer:     @staticmethod     def play_quiz():         print("\n--- Rozpoczęcie odtwarzania quizu ---") # Informacja o trybie         # ...         print("Dostępne quizy:") # Nagłówek listy quizów         for i, quiz_name in enumerate(available_quizzes):             print(f" {i + 1}. {quiz_name}") # Wyświetlanie nazw quizów         # ...         print(f"\n--- Rozpoczęcie quizu: {quiz.title} ---") # Informacja o rozpoczęciu quizu         # ...         print("Poprawna odpowiedź!") # Informacja o poprawnej odpowiedzi         # ...         print(f"Niepoprawna odpowiedź. Poprawna to: {question.options[question.correct_answer_index]}") # Informacja o błędnej odpowiedzi         # ...         print("\n--- Koniec quizu! ---") # Zakończenie quizu         print(f"Twój wynik: {correct_answers_count}/{total_questions} poprawnych odpowiedzi.") # Podsumowanie wyniku      @staticmethod     def generate_and_save_results_chart(correct_count: int, incorrect_count: int, quiz_title: str):         # ...         print(f"Wykres wyników został zapisany w: {chart_filepath}") # Potwierdzenie zapisu wykresu </pre>			
--	--	--	--	--

	Funkcje z parametrami i wartościami zwracanymi	<pre># models/question.py (fragmenty)  class Question:     # ... (kod wcześniejszy) ...      def is_correct(self, user_answer_index: int) -&gt; bool:         """         Checks if the user's provided answer index matches         the correct answer.          Args:             user_answer_index (int): The 0-based index of             the user's chosen answer.          Returns:             bool: True if the answer is correct, False             otherwise.         """         return user_answer_index == self.correct_answer_index  # quiz_data/manager.py (fragmenty)  class QuizDataManager:     # ... (kod wcześniejszy) ...      @staticmethod     def list_available_quizzes(directory: str = "data/quiz_examples") -&gt; list[str]:         """         Lists all available quiz files (JSON files) in the         specified directory.          Args:             directory (str): The directory to search for quiz             files.              Defaults to "data/quiz_examples".          Returns:             list[str]: A list of filenames (without the .json             extension)</pre>		2	2
--	--	--	---	---	---

	<pre>         representing available quizzes. Returns an         empty list         if the directory does not exist or contains         no quizzes.         """         if not os.path.exists(directory):             return []          quiz_files = []         try:             for item in os.listdir(directory):                 if item.endswith(".json"):                     quiz_files.append(os.path.splitext(item)[0])             quiz_files.sort()         except OSError as e:             print(f"Error listing files in directory {directory}:             {e}")         return quiz_files </pre>			
Funkcje rekurencyjne	<pre> def select_unique_questions_recursive(     all_questions: list,     num_to_select: int,     _selected_indices: set = None, # Zbiór indeksów już     wybranych pytań     _result_questions: list = None # Lista pytań do     zwrócenia ) -&gt; list:     """     Recursively selects a specified number of unique     questions from a list.      Note: While this demonstrates recursion, for practical     purposes,     an iterative approach using random.sample or a loop     with a set     is often more efficient and less prone to stack     overflow for large N.      Args:         all_questions (list): The list of all available question         objects. </pre>		3	3


	<p>num_to_select (int): The number of unique questions to select.</p> <p>_selected_indices (set, optional): Internal parameter, set of indices of questions already selected in current path.</p> <p>Defaults to None (initialized on first call).</p> <p>_result_questions (list, optional): Internal parameter, list of question objects selected so far. Defaults to None.</p> <p>Returns:</p> <p>list: A list of unique question objects.</p> <p>Raises:</p> <p>ValueError: If num_to_select is negative or greater than available questions.</p> <pre> """ if _selected_indices is None:     _selected_indices = set() if _result_questions is None:     _result_questions = []  if not isinstance(num_to_select, int) or num_to_select &lt; 0:     raise ValueError("Number of questions to select must be a non-negative integer.") if num_to_select &gt; len(all_questions):     raise ValueError("Cannot select more questions than available.")  # Base Case 1: If we have selected enough questions if len(_result_questions) == num_to_select:     return _result_questions  # Base Case 2: If no more unique questions are available but we still need more if len(_selected_indices) == len(all_questions): </pre>			
--	--	--	--	--


	<pre> # We've exhausted all questions but haven't met num_to_select # This can happen if num_to_select was greater than actual unique questions return _result_questions # Return what we could select  # Recursive Step: Select a random unique question available_to_pick_indices = [i for i in range(len(all_questions)) if i not in _selected_indices]  if not available_to_pick_indices: # No more unselected questions (should be caught by Base Case 2, but for robustness) return _result_questions  # Randomly pick one index from the remaining available questions chosen_index = random.choice(available_to_pick_indices)  # Add the chosen index to the set of selected indices _selected_indices.add(chosen_index)  # Add the corresponding question object to our result list  _result_questions.append(all_questions[chosen_index])  # Recursive call: try to select one less question return select_unique_questions_recursive(     all_questions,     num_to_select,     _selected_indices=_selected_indices,     _result_questions=_result_questions ) </pre>			
Funkcje przyjmujące inne funkcje	<pre> # quiz_project/quiz_player/player.py (fragment z metody QuizPlayer.play_quiz)  # ... (kod wcześniejszy - zbieranie odpowiedzi do user_answers) ... </pre>		3	3


		jako argumenty	<pre> # --- Analiza wyników z użyciem programowania funkcyjnego ---  # Filter dla poprawnych odpowiedzi # Funkcja 'filter' przyjmuje funkcję (lambda) jako pierwszy argument correct_results = list(filter(lambda ans: ans["is_correct"], user_answers)) num_correct = len(correct_results)  # Filter dla niepoprawnych odpowiedzi # Funkcja 'filter' przyjmuje funkcję (lambda) jako pierwszy argument incorrect_results = list(filter(lambda ans: not ans["is_correct"], user_answers)) num_incorrect = len(incorrect_results)  # Użycie map do pobrania treści pytań, na które odpowiedziano błędnie # Funkcja 'map' przyjmuje funkcję (lambda) jako pierwszy argument incorrect_question_texts = list(map(lambda ans: ans["question_text"], incorrect_results)) if incorrect_question_texts:     print("\nPytań, na które odpowiedziałeś/aś błędnie:")     for text in incorrect_question_texts:         print(f"- {text}")  # ... (kod późniejszy) ... </pre>			
		Dekoratory	<pre> from utils.helpers import timing_decorator  # ... class QuizPlayer:     @staticmethod     @timing_decorator     def play_quiz():         # ... (cała logika odtwarzania quizu) ... </pre>		1.5	1, 5
3	Kontener y	Użycie listy	<pre> # models/question.py (fragment z konstruktora Question) </pre>		2	2


			<pre> class Question:     def __init__(self, question_text: str, options: list, correct_answer_index: int):         # Lista do przechowywania opcji odpowiedzi pytania         self.options = [opt.strip() for opt in options]  # models/quiz.py (fragment z konstruktora Quiz i metody add_question)  class Quiz:     def __init__(self, title: str, description: str = "", questions: list = None):         # Lista do przechowywania obiektów Question w quizie         self.questions = []         if questions:             for q in questions:                 self.questions.append(q)      def add_question(self, question: Question):         # Dodawanie elementu do listy         self.questions.append(question)  # quiz_player/player.py (fragment z metody play_quiz)  class QuizPlayer:     @staticmethod     def play_quiz():         # Lista do przechowywania nazw dostępnych quizów         available_quizzes = QuizDataManager.list_available_quizzes()          # Lista do przechowywania odpowiedzi użytkownika         user_answers = []         # ...         # Dodawanie słownika reprezentującego odpowiedź do listy </pre>			
--	--	--	---	--	--	--



		<pre> user_answers.append({     "question_text": question.question_text,     "user_choice_index": answer_index,     "is_correct": question.is_correct(answer_index),     "correct_answer_index": question.correct_answer_index,     "options": question.options }) </pre>			
	Użycie słownika	<pre> # models/question.py (fragmenty z metod to_dict i from_dict)  class Question:     # ... (inne metody) ...      def to_dict(self) -&gt; dict:         """         Converts the Question object to a dictionary for         JSON serialization.         Returns:             dict: A dictionary representation of the question.         """         # Słownik do reprezentacji obiektu Question w         formacie JSON         return {             "question_text": self.question_text,             "options": self.options,             "correct_answer_index": self.correct_answer_index         }      @classmethod     def from_dict(cls, data: dict):         """         Creates a Question object from a dictionary (for         JSON deserialization).         Args:             data (dict): A dictionary containing question             data.         """         # Przyjmowanie słownika jako danych wejściowych         return cls( </pre>		2	2


	<pre> question_text=data["question_text"], options=data["options"],  correct_answer_index=data["correct_answer_index"] )  # quiz_player/player.py (fragment z metody QuizPlayer.play_quiz)  class QuizPlayer:     @staticmethod     def play_quiz():         # ... (kod wcześniejszy) ...         user_answers = []         # ...          # Słownik do przechowywania szczegółów         odpowiedzi użytkownika na każde pytanie         user_answers.append({             "question_text": question.question_text,             "user_choice_index": answer_index,             "is_correct": question.is_correct(answer_index),             "correct_answer_index": question.correct_answer_index,             "options": question.options         })         # ... (kod późniejszy) ... </pre>			
Użycie zbioru	<pre> # quiz_project/utils/helpers.py (fragment z funkcji select_unique_questions_recursive)  def select_unique_questions_recursive(     all_questions: list,     num_to_select: int,     _selected_indices: set = None, # Zbiór indeksów już wybranych pytań     _result_questions: list = None ) -&gt; list:     """     Recursively selects a specified number of unique     questions from a list.     """ </pre>		1.5	1, 5


		<pre> if _selected_indices is None:     _selected_indices = set() # Inicjalizacja pustego zbioru  # ... (kod wcześniejszy) ...  # Krok rekurencyjny: Wybierz losowe, unikalne pytanie available_to_pick_indices = [i for i in range(len(all_questions)) if i not in _selected_indices]  # ... (logika wyboru losowego indeksu) ...  # Dodaj wybrany indeks do zbioru już wybranych indeksów, aby zapewnić unikalność _selected_indices.add(chosen_index)  # ... (kod późniejszy) ... </pre>			
	Użycie krotki	<pre> @staticmethod def generate_and_save_results_chart(correct_count: int, incorrect_count: int, quiz_title: str):     """     Generates a pie chart showing the distribution of     correct vs. incorrect answers     and saves it to the 'reports' directory.      Args:         correct_count (int): Number of correct answers.         incorrect_count (int): Number of incorrect         answers.         quiz_title (str): The title of the quiz for chart         labeling.     """     # Użycie krotki do przechowywania stałych etykiet     labels: tuple[str, str] = ('Poprawne', 'Niepoprawne')     sizes = [correct_count, incorrect_count]      # Użycie krotki do przechowywania stałych kodów     kolorów     colors: tuple[str, str] = ('#4CAF50', '#F44336') #     Green for correct, Red for incorrect </pre>		1.5	1, 5

			<pre> explode = (0.1, 0) # explode the 1st slice (Correct)  fig1, ax1 = plt.subplots() ax1.pie(sizes, explode=explode, labels=labels, colors=colors,         autopct='%1.1f%%', shadow=True, startangle=90, textprops={'fontsize': 12, 'color': 'white'}) ax1.axis('equal') # Equal aspect ratio ensures that pie is drawn as a circle.  plt.title(f'Wyniki quizu: {quiz_title}', fontsize=16, color='black')  # Save the chart reports_dir = "reports" if not os.path.exists(reports_dir):     os.makedirs(reports_dir)     print(f"Created directory: {reports_dir}")  chart_filename = f"wyniki_{quiz_title.replace(' ', '_' ).lower()}.png" chart_filepath = os.path.join(reports_dir, chart_filename)  try:     plt.savefig(chart_filepath, bbox_inches='tight', dpi=100)     print(f"Wykres wyników został zapisany w: {chart_filepath}") except Exception as e:     print(f"Błąd podczas zapisywania wykresu: {e}") finally:     plt.close(fig1) # Close the plot to free up memory </pre>			
4	Przestrze nie nazw	Zastosowan o zmienne lokalne	<pre> # quiz_project/quiz_creator/creator.py (fragment z metody QuizCreator.create_new_quiz)  class QuizCreator:     @staticmethod     def create_new_quiz(): </pre>		1.5	1, 5


			<pre> # 'title' jest zmienną lokalną, dostępną tylko wewnątrz create_new_quiz title = input("Podaj tytuł quizu (np. 'Geografia Polski'): ").strip()  # 'description' jest zmienną lokalną description = input("Podaj krótki opis quizu (opcjonalnie): ").strip()  # 'new_quiz' jest zmienną lokalną, przechowującą obiekt Quiz new_quiz = Quiz(title, description)  # 'question_text' jest zmienną lokalną, dostępną w pętli dodawania pytań question_text = input("Wpisz treść pytania...").strip()  # 'options' jest zmienną lokalną (lista) options = []  # 'option_count' jest zmienną lokalną option_count = 1  # 'filename' jest zmienną lokalną filename = input("Podaj nazwę pliku...").strip()  # quiz_project/quiz_player/player.py (fragment z metody QuizPlayer.play_quiz)  class QuizPlayer:     @staticmethod     def play_quiz():         # 'available_quizzes' jest zmienną lokalną         available_quizzes = QuizDataManager.list_available_quizzes()  # 'selected_quiz_name' jest zmienną lokalną selected_quiz_name = None </pre>			
--	--	--	--	--	--	--

			<pre> # 'quiz' jest zmienną lokalną, przechowującą wczytany obiekt Quiz quiz = None  # 'user_answers' jest zmienną lokalną (lista słowników) user_answers = []  # 'correct_answers_count' i 'total_questions' to zmienne lokalne correct_answers_count = 0 total_questions = len(quiz.questions)  # 'choice' i 'choice_index' to zmienne lokalne, używane do interakcji z użytkownikiem choice = input("Wybierz numer quizu...").strip() choice_index = int(choice) - 1  # 'num_correct' i 'num_incorrect' to zmienne lokalne num_correct = len(correct_results) num_incorrect = len(incorrect_results)  # quiz_project/quiz_player/player.py (fragment z metody QuizPlayer.generate_and_save_results_chart)  @staticmethod def generate_and_save_results_chart(correct_count: int, incorrect_count: int, quiz_title: str):     # 'labels', 'sizes', 'colors', 'explode', 'fig1', 'ax1', 'reports_dir', 'chart_filename', 'chart_filepath'     # to wszystko są zmienne lokalne tej metody     labels: tuple[str, str] = ('Poprawne', 'Niepoprawne')     sizes = [correct_count, incorrect_count]     colors: tuple[str, str] = ('#4CAF50', '#F44336')     explode = (0.1, 0)     fig1, ax1 = plt.subplots()     reports_dir = "reports"     chart_filename = f"wyniki_{quiz_title.replace(' ', '_').lower()}.png" </pre>			
--	--	--	---	--	--	--


			<code>chart_filepath = os.path.join(reports_dir, chart_filename)</code>			
	Zastosowa o zmienne globalne		<code># quiz_project/quiz_player/player.py (fragment)</code>  <code>import matplotlib</code> <code>matplotlib.use('Agg') # Ustawia nieinteraktywny</code> <code>backend, aby zapobiec problemom z GUI w</code> <code>testach/środowiskach bez GUI</code> <code>import matplotlib.pyplot as plt</code> <code>import os</code> <code>from models.quiz import Quiz</code> <code>from quiz_data.manager import QuizDataManager</code>  <code># Zmienna globalna na poziomie modułu</code> <code># Jest dostępna dla wszystkich funkcji i metod w tym</code> <code>pliku.</code> <code>REPORTS_DIRECTORY = "reports"</code>  <code>class QuizPlayer:</code> <code>    # ... (inne metody i kod) ...</code>  <code>@staticmethod</code> <code>def generate_and_save_results_chart(correct_count:</code> <code>int, incorrect_count: int, quiz_title: str):</code> <code>    """</code> <code>        Generates a pie chart showing the distribution of</code> <code>correct vs. incorrect answers</code> <code>and saves it to the 'reports' directory.</code> <code>    """</code> <code>    # ... (kod generujący wykres) ...</code>  <code># Użycie globalnej zmiennej REPORTS_DIRECTORY</code> <code>if not os.path.exists(REPORTS_DIRECTORY):</code> <code>    os.makedirs(REPORTS_DIRECTORY)</code> <code>print(f"Created directory:</code> <code>{REPORTS_DIRECTORY}")</code>  <code>chart_filename = f"wyniki_{quiz_title.replace(' ',</code> <code>'_').lower()}.png"</code> <code># Użycie globalnej zmiennej REPORTS_DIRECTORY</code>		1.5	
						1, 5

	<pre> chart_filepath = os.path.join(REPORTS_DIRECTORY, chart_filename)  # ... (dalszy kod zapisu) ... </pre>			
Zastosowa o zakresy funkcji	<pre> # quiz_project/quiz_creator/creator.py (fragmenty z metod create_new_quiz i _add_questions_to_quiz)  class QuizCreator:     @staticmethod     def create_new_quiz():         # Zmienna 'title' jest lokalna dla funkcji create_new_quiz         title = input("Podaj tytuł quizu...").strip()          # Zmienna 'description' jest lokalna dla funkcji create_new_quiz         description = input("Podaj krótki opis...").strip()          new_quiz = Quiz(title, description) # 'new_quiz' również jest lokalna          # Wywołanie innej funkcji/metody QuizCreator._add_questions_to_quiz(new_quiz)          # Zmienne 'title', 'description', 'new_quiz' są dostępne tylko tutaj         # Zmienne z _add_questions_to_quiz NIE SĄ dostępne bezpośrednio tutaj      @staticmethod     def _add_questions_to_quiz(quiz: Quiz):         # 'question_text' jest lokalna dla funkcji _add_questions_to_quiz         question_text = input("Wpisz treść pytania...").strip()          # 'options' jest lokalna dla funkcji _add_questions_to_quiz         options = []          # 'option_count' jest lokalna dla funkcji _add_questions_to_quiz </pre>		1.5	1, 5





		<pre> option_count = 1  # 'option' jest lokalna dla wewnętrznej pętli while, # ale dostępna w zakresie _add_questions_to_quiz while True:     option = input(f"Opcja {option_count}: ").strip()     # ...     if not option:         break     # ...  # 'correct_answer_index' jest lokalna dla _add_questions_to_quiz correct_answer_index = -1 # ... # Zmienne 'question_text', 'options', 'option_count', 'correct_answer_index' # są dostępne tylko wewnątrz funkcji _add_questions_to_quiz. # NIE SĄ dostępne bezpośrednio w create_new_quiz. </pre>			
	Zastosowa no zakresy klas	<pre> # models/question.py (fragment)  class Question:     """     Represents a single question in a quiz. # Ten     docstring jest częścią zakresu klasy     """      # Atrybuty instancji, które są dostępne w zakresie     klasy poprzez self.     question_text: str     options: list     correct_answer_index: int      def __init__(self, question_text: str, options: list,     correct_answer_index: int):         # self.question_text odwołuje się do atrybutu         instancji w zakresie klasy         self.question_text = question_text.strip()         self.options = [opt.strip() for opt in options]         self.correct_answer_index = correct_answer_index </pre>		1.5	1, 5


		<pre> def is_correct(self, user_answer_index: int) -&gt; bool:     # is_correct to metoda dostępna w zakresie klasy     Question     # Odwołuje się do self.correct_answer_index, który     jest atrybutem instancji     return user_answer_index ==     self.correct_answer_index  @classmethod def from_dict(cls, data: dict):     # from_dict to metoda klasy, również w zakresie     klasy Question     # cls odwołuje się do samej klasy Question     return cls(         question_text=data["question_text"],         options=data["options"],          correct_answer_index=data["correct_answer_index"]     )  # models/quiz.py (fragment)  class Quiz:     """     Represents a collection of questions that form a quiz.     # Docstring dla klasy Quiz     """     # Atrybuty instancji quizu, dostępne w zakresie klasy     title: str     description: str     questions: list      def __init__(self, title: str, description: str = "",     questions: list = None):         self.title = title.strip() # self.title odwołuje się do         atrybutu instancji         self.questions = [] # self.questions to lista w         zakresie tej klasy      def add_question(self, question: Question): </pre>			
--	--	---	--	--	--

			<pre># add_question to metoda klasy, dostępna w jej zakresie self.questions.append(question) # Modyfikuje self.questions</pre>			
5	Moduły i pakiety	Projekt podzielony na moduły (import, __init__)	<pre># Przykładowa struktura katalogów i plików w projekcie: # quiz_project/ #  — main.py #  — models/ #      — __init__.py # Inicjalizuje 'models' jako pakiet #      — question.py #     └─ quiz.py #  — quiz_data/ #      — __init__.py # Inicjalizuje 'quiz_data' jako pakiet #     └─ manager.py #  — quiz_creator/ #      — __init__.py # Inicjalizuje 'quiz_creator' jako pakiet #     └─ creator.py #  — quiz_player/ #      — __init__.py # Inicjalizuje 'quiz_player' jako pakiet #     └─ player.py # └─ utils/ #      — __init__.py # Inicjalizuje 'utils' jako pakiet #     └─ helpers.py  # Przykład pliku __init__.py (może być pusty, ale musi istnieć) # quiz_project/models/__init__.py # (Ten plik jest zazwyczaj pusty, ale jego obecność sprawia, że katalog 'models' # jest traktowany jako pakiet Pythona, co pozwala na importowanie z niego modułów)  # Przykłady instrukcji import w projekcie:  # Z main.py importujemy z pakietów # quiz_project/main.py (fragment) import sys</pre>		2	2

		<pre> import os # ... from quiz_creator.creator import QuizCreator # Import z pakietu quiz_creator from quiz_player.player import QuizPlayer # Import z pakietu quiz_player from quiz_data.manager import QuizDataManager # Import z pakietu quiz_data  # Z quiz_data/manager.py importujemy z pakietu models # quiz_project/quiz_data/manager.py (fragment) import json import os from models.question import Question # Import klasy Question z pakietu models from models.quiz import Quiz # Import klasy Quiz z pakietu models  # Z models/quiz.py importujemy z tego samego pakietu # quiz_project/models/quiz.py (fragment) from .question import Question # Import Question z bieżącego pakietu models # ... (reszta kodu) ...  # Z tests/test_interactive_modules.py importujemy z różnych pakietów projektu # quiz_project/tests/test_interactive_modules.py (fragment) import unittest import sys import os # ... from models.question import Question # Import z pakietu models from models.quiz import Quiz # Import z pakietu models from quiz_data.manager import QuizDataManager # Import z pakietu quiz_data </pre>		
--	--	---	--	--


			from quiz_creator.creator import QuizCreator # Import z pakietu quiz_creator from quiz_player.player import QuizPlayer # Import z pakietu quiz_player			
Nr	Obszar	Wymaganie	KOD		Przyznane pkt	Pkt max
		Własne pakiety/funkcje pomocnicze w osobnych plikach .py	<pre># quiz_project/utils/helpers.py  import random import time from functools import wraps  def factorial_recursive(n: int) -&gt; int:     """     Calculates the factorial of a non-negative integer     using recursion.     This is a custom helper function.     """     if not isinstance(n, int):         raise TypeError("Input must be an integer.")     if n &lt; 0:         raise ValueError("Factorial is not defined for negative numbers.")     if n == 0:         return 1     else:         return n * factorial_recursive(n - 1)  def timing_decorator(func):     """     A decorator that measures the execution time of a     function.     This is a custom helper function/decorator.     """     @wraps(func)     def wrapper(*args, **kwargs):         start_time = time.perf_counter()         result = func(*args, **kwargs)</pre>		2	2

			<pre> end_time = time.perf_counter() execution_time = end_time - start_time print(f"[{func.__name__}] Czas wykonania: {execution_time:.4f} sekund") return result return wrapper  # ... (inne funkcje pomocnicze, np. select_unique_questions_recursive) ... </pre>			
6	Obsługa błędów	Obsługa wyjątków (try, except, finally)	<pre> # quiz_project/quiz_data/manager.py (fragment metody QuizDataManager.load_quiz)  class QuizDataManager:     # ... (inne metody) ...      @staticmethod     def load_quiz(filename: str, directory: str = "data/quiz_examples") -&gt; Quiz:     """     Loads a Quiz object from a JSON file.     """     if not filename.endswith(".json"):         filename += ".json"      file_path = os.path.join(directory, filename)      if not os.path.exists(file_path):         # Warunek wstępny, rzuca FileNotFoundError jeśli plik nie istnieje         raise FileNotFoundError(f"Quiz file not found: {file_path}")      try: # Blok 'try': kod, który może wygenerować wyjątek         with open(file_path, 'r', encoding='utf-8') as f:             quiz_data = json.load(f) # json.load() może rzucić JSONDecodeError             quiz = Quiz.from_dict(quiz_data) # from_dict() może rzucić KeyError             print(f"Quiz '{quiz.title}' loaded successfully from {file_path}") </pre>		2	2


		<pre> return quiz except json.JSONDecodeError as e: # Blok 'except': obstuga konkretnego wyjątku JSONDecodeError     print(f"Error decoding JSON from {file_path}: {e}")     raise json.JSONDecodeError(f"Invalid JSON format in file: {file_path}", e.doc, e.pos) from e except KeyError as e: # Blok 'except': obstuga konkretnego wyjątku KeyError     print(f"Missing required data in quiz file {file_path}: {e}")     raise KeyError(f"Corrupt quiz file. Missing key: {e}") from e except Exception as e: # Ogólny blok 'except': obstuga innych nieoczekiwanych wyjątków     print(f"An unexpected error occurred while loading quiz from {file_path}: {e}")     raise finally: # Blok 'finally': kod, który zawsze zostanie wykonany, niezależnie od wyjątku     # W tym przypadku, użycie 'with open' już zapewnia zamknięcie pliku,     # ale 'finally' może być użyte np. do zwolnienia innych zasobów     pass # Brak konkretnych zasobów do zwolnienia poza plikiem </pre>			
	Użycie assert do testów i walidacji	<pre> # quiz_project/tests/test_models.py (fragmenty z klasy TestQuestion i TestQuiz)  import unittest # ... (inne importy) ...  class TestQuestion(unittest.TestCase):     """Unit tests for the Question class."""      def test_question_initialization_valid(self):         """Test Question initialization with valid data."""         question = Question("What is 2+2?", ["3", "4", "5"], 1)          # Użycie assertEquals do sprawdzenia równości atrybutów </pre>		1.5	1, 5

		<pre> self.assertEqual(question.question_text, "What is 2+2?") self.assertEqual(question.options, ["3", "4", "5"]) self.assertEqual(question.correct_answer_index, 1)  def test_question_initialization_empty_text_raises_error(self):     """Test Question initialization with empty question     text."""     # Użycie assertRaisesRegex do sprawdzenia, czy     rzucany jest wyjątek ValueError     with self.assertRaisesRegex(ValueError, "Question     text cannot be empty."):         Question("", ["a", "b"], 0)      def test_question_is_correct(self):         """Test the is_correct method for correct and         incorrect answers."""         question = Question("What is 2+2?", ["3", "4", "5"],         1)         # Użycie assertTrue/assertFalse do sprawdzenia         wartości logicznych         self.assertTrue(question.is_correct(1)) # Oczekujemy         True         self.assertFalse(question.is_correct(0)) #         Oczekujemy False      def test_question_to_dict_from_dict(self):         """Test conversion to dictionary and back to         object."""         original_question = Question("Fav color?", ["Red",         "Blue"], 0)         q_dict = original_question.to_dict()          # Użycie isinstance do sprawdzenia typu         obiektu         self.assertIsInstance(q_dict, dict)         # Użycie assertEquals do sprawdzenia wartości         self.assertEqual(q_dict["question_text"], "Fav         color?") </pre>			
--	--	---	--	--	--





			<pre> reconstructed_question = Question.from_dict(q_dict) self.assertIsInstance(reconstructed_question, Question)  self.assertEqual(reconstructed_question.question_text, original_question.question_text)  class TestQuiz(unittest.TestCase):     """Unit tests for the Quiz class."""     # ... (kod setUp) ...      def test_add_question(self):         """Test adding questions to the quiz."""         quiz = Quiz("Test Add")         quiz.add_question(self.q1)         # Użycie assertEquals do sprawdzenia rozmiaru listy         self.assertEqual(len(quiz.questions), 1)         # Użycie assertEquals do sprawdzenia treści pytania         self.assertEqual(quiz.questions[0].question_text, "Q1 text")      def test_remove_question_invalid_index_raises_error(self):         """Test removing question with out-of-bounds index."""         quiz = Quiz("Test Remove Invalid", questions=[self.q1])         # Użycie assertRaisesRegex do sprawdzenia, czy rzucany jest wyjątek IndexError         with self.assertRaisesRegex(IndexError, "Question index is out of bounds."):             quiz.remove_question(1) </pre>			
7	łańcuchy znaków	Operacje na stringach (m.in. formatowanie, dzielenie,	<pre> # models/question.py (fragment z konstruktora __init__)  class Question:     def __init__(self, question_text: str, options: list, correct_answer_index: int):         # Operacja czyszczenia: usunięcie białych znaków z początku i końca stringa </pre>		2	2


		wyszukiwa nie)	<pre> self.question_text = question_text.strip() self.options = [opt.strip() for opt in options]  # quiz_creator/creator.py (fragment z metod create_new_quiz i _save_quiz_with_prompt)  class QuizCreator:     @staticmethod     def create_new_quiz():         # Operacja czyszczenia: usunięcie białych znaków z         inputu         title = input("Podaj tytuł quizu...").strip()         description = input("Podaj krótki opis...").strip()          # Operacja formatowania: f-string do         dynamicznego tworzenia komunikatu         print(f"Quiz '{title}' został utworzony. Teraz dodaj         pytania.")      @staticmethod     def _save_quiz_with_prompt(quiz: Quiz):         # Operacja wyszukiwania (sprawdzenie sufiksu): czy         nazwa pliku kończy się na ".json"         if not filename.endswith(".json"):             filename += ".json" # Operacja konkatencji             stringów          # Operacja formatowania: f-string do         dynamicznego tworzenia komunikatu         full_path = os.path.join("data/quiz_examples",         filename + ".json")         overwrite = input(f"Plik '{filename}.json' już istnieje.         Czy chcesz go nadpisać? (tak/nie: ").lower()         # Operacja modyfikacji: zmiana wszystkich liter na         małe         # Operacja porównania: sprawdzanie czy string jest         równy "tak"         if overwrite != 'tak':             print("Zapisywanie quizu anulowane.") </pre>			
--	--	-------------------	---	--	--	--

			<pre> # quiz_player/player.py (fragment z metody generate_and_save_results_chart)  class QuizPlayer:     @staticmethod     def generate_and_save_results_chart(correct_count: int, incorrect_count: int, quiz_title: str):         # Operacje modyfikacji: zamiana spacji na podkreślenia, zmiana na małe litery         chart_filename = f"wyniki_{quiz_title.replace(' ', ' ').lower()}.png"         # Operacja formatowania: f-string do dynamicznego tworzenia komunikatu o ścieżce         print(f"Wykres wyników został zapisany w: {chart_filepath}")  # quiz_data/manager.py (fragment z metody list_available_quizzes)  class QuizDataManager:     @staticmethod     def list_available_quizzes(directory: str = "data/quiz_examples") -&gt; list[str]:         # Operacja wyszukiwania (sprawdzenie sufiksu): czy element kończy się na ".json"         # Operacja dzielenia (niejawna): os.path.splitext() dzieli nazwę pliku na nazwę i rozszerzenie         for item in os.listdir(directory):             if item.endswith(".json"):                 quiz_files.append(os.path.splitext(item)[0]) # [0] pobiera pierwszą część (nazwę bez rozszerzenia) </pre>			
8	Obsługa plików	Odczyt z plików .txt, .csv, .json, .xml (min. 1)	<pre> # quiz_project/quiz_data/manager.py (fragment metody QuizDataManager.load_quiz)  class QuizDataManager:     # ... (inne metody) ...      @staticmethod </pre>		2	2

			<pre> def load_quiz(filename: str, directory: str = "data/quiz_examples") -&gt; Quiz:     """     Loads a Quiz object from a JSON file.     """     if not filename.endswith(".json"):         filename += ".json"      file_path = os.path.join(directory, filename)      if not os.path.exists(file_path):         # Obsługa błędu: plik nie istnieje         raise FileNotFoundError(f"Quiz file not found: {file_path}")      try:         with open(file_path, 'r', encoding='utf-8') as f:             # Odczyt danych z pliku JSON             quiz_data = json.load(f)             quiz = Quiz.from_dict(quiz_data)             print(f"Quiz '{quiz.title}' loaded successfully from {file_path}")         return quiz     except json.JSONDecodeError as e:         # Obsługa błędu: nieprawidłowy format JSON         print(f"Error decoding JSON from {file_path}: {e}")         raise json.JSONDecodeError(f"Invalid JSON format in file: {file_path}", e.doc, e.pos) from e     except KeyError as e:         # Obsługa błędu: brakujące klucze w strukturze JSON         print(f"Missing required data in quiz file {file_path}: {e}")         raise KeyError(f"Corrupt quiz file. Missing key: {e}") from e     except Exception as e:         # Obsługa innych nieoczekiwanych błędów         print(f"An unexpected error occurred while loading quiz from {file_path}: {e}")         raise </pre>			
--	--	--	--	--	--	--


	<p>Zapis do plików .txt, .csv, .json, .xml (min. 1)</p>	<pre># quiz_project/quiz_data/manager.py (fragment metody QuizDataManager.save_quiz)  class QuizDataManager:     # ... (inne metody) ...      @staticmethod     def save_quiz(quiz: Quiz, filename: str, directory: str = "data/quiz_examples"):         """         Saves a Quiz object to a JSON file.          Args:             quiz (Quiz): The Quiz object to be saved.             filename (str): The name of the file (e.g., "my_quiz.json").             directory (str): The directory where the quiz file will be saved.              Defaults to "data/quiz_examples".          Raises:             IOError: If there's an issue writing to the file (e.g., permissions).             TypeError: If the provided object is not a Quiz instance.         """         if not isinstance(quiz, Quiz):             raise TypeError("Only Quiz objects can be saved.")         if not filename.endswith(".json"):             filename += ".json" # Ensure filename has .json extension          # Ensure the directory exists         if not os.path.exists(directory):             try:                 os.makedirs(directory)                 print(f"Created directory: {directory}")             except OSError as e:                 print(f"Error creating directory {directory}: {e}")</pre>		2	
					2


			<pre>         raise IOError(f"Could not create directory {directory}.") from e          file_path = os.path.join(directory, filename)          try:             # Convert Quiz object to a dictionary             quiz_data = quiz.to_dict()             with open(file_path, 'w', encoding='utf-8') as f:                 # Zapis danych do pliku JSON                 json.dump(quiz_data, f, indent=4, ensure_ascii=False)             print(f"Quiz '{quiz.title}' saved successfully to {file_path}")         except IOError as e:             print(f"Error saving quiz to {file_path}: {e}")             raise IOError(f"Failed to write quiz to file: {file_path}") from e         except Exception as e: # Catch any other unexpected errors during serialization             print(f"An unexpected error occurred while saving quiz: {e}")         raise </pre>			
9	OOP	Klasy	<pre> # models/question.py (fragment definicji klasy Question)  class Question:     """     Represents a single question in a quiz.      Attributes:         question_text (str): The text of the question.         options (list): A list of strings, where each string is a possible answer option.         correct_answer_index (int): The 0-based index of the correct answer in the 'options' list.     """     # ... (definicja konstruktora i metod) ...  # models/quiz.py (fragment definicji klasy Quiz) </pre>		2	2

		<pre> class Quiz:     """     Represents a collection of questions that form a quiz.      Attributes:         title (str): The title of the quiz.         description (str): An optional description of the         quiz.         questions (list): A list of Question objects belonging         to this quiz.     """     # ... (definicja konstruktora i metod) ... </pre>			
	Metody	<pre> # models/question.py (fragmenty)  class Question:     # ... (definicja konstruktora) ...      def display(self) -&gt; str:         """         Returns a formatted string for displaying the         question and its options.         To jest metoda instancji, która formatuje i zwraca         tekst pytania.         """         display_str = f"Pytanie: {self.question_text}\n"         for i, option in enumerate(self.options):             display_str += f" {i + 1}. {option}\n"         return display_str      def is_correct(self, user_answer_index: int) -&gt; bool:         """         Checks if the user's provided answer index matches         the correct answer.         To jest metoda instancji, która porównuje dane         użytkownika z danymi obiektu.         """         return user_answer_index ==         self.correct_answer_index      def to_dict(self) -&gt; dict: </pre>		2	2


	<pre> """     Converts the Question object to a dictionary for     JSON serialization.     To jest metoda instancji, która transformuje stan     obiektu.     """     return {         "question_text": self.question_text,         "options": self.options,         "correct_answer_index": self.correct_answer_index     }  # models/quiz.py (fragmenty)  class Quiz:     # ... (definicja konstruktora) ...      def add_question(self, question: 'Question'):         """         Adds a Question object to the quiz.         To jest metoda instancji, która modyfikuje atrybut         obiektu (listę pytań).         """         if not isinstance(question, Question):             raise TypeError("Only Question objects can be added to the quiz.")         self.questions.append(question)      def remove_question(self, index: int):         """         Removes a question from the quiz by its index.         To jest metoda instancji, która modyfikuje atrybut         obiektu.         """         if not isinstance(index, int):             raise TypeError("Index must be an integer.")         if not (0 &lt;= index &lt; len(self.questions)):             raise IndexError("Question index is out of bounds.")         self.questions.pop(index) </pre>			
--	---	--	--	--






		<p>Konstruktor y</p> <pre># models/question.py (fragment definicji klasy Question)  class Question:     """     Represents a single question in a quiz.     """     def __init__(self, question_text: str, options: list, correct_answer_index: int):         """         Initializes a new Question object.         To jest konstruktor klasy Question.         Przyjmuje parametry i używa ich do ustawienia początkowych wartości atrybutów obiektu.         """         # Walidacja danych wejściowych         if not isinstance(question_text, str) or not question_text.strip():             raise ValueError("Question text cannot be empty.")         if not isinstance(options, list) or not options:             raise ValueError("Options must be a non-empty list.")         if not all(isinstance(opt, str) and opt.strip() for opt in options):             raise ValueError("All options must be non-empty strings.")         if not isinstance(correct_answer_index, int) or not (0 &lt;= correct_answer_index &lt; len(options)):             raise ValueError("Correct answer index is out of bounds or not an integer.")          # Inicjalizacja atrybutów instancji         self.question_text = question_text.strip()         self.options = [opt.strip() for opt in options]         self.correct_answer_index = correct_answer_index          # ... (inne metody) ...  # models/quiz.py (fragment definicji klasy Quiz)</pre>		2	2
--	--	---	---	---	---


		<pre> class Quiz:     """     Represents a collection of questions that form a quiz.     """      def __init__(self, title: str, description: str = "", questions: list = None):         """         Initializes a new Quiz object.         To jest konstruktor klasy Quiz.         Ustawia tytuł, opis i listę pytań dla nowego obiektu Quiz.         """          # Walidacja danych wejściowych         if not isinstance(title, str) or not title.strip():             raise ValueError("Quiz title cannot be empty.")          # Inicjalizacja atrybutów instancji         self.title = title.strip()         self.description = description.strip() if description else ""         self.questions = []         if questions:             for q in questions:                 if not isinstance(q, Question):                     raise TypeError("All items in 'questions' must be Question objects.")                 self.questions.append(q)          # ... (inne metody) ... </pre>			
	Dziedziczenie	<pre> # quiz_project/tests/test_models.py (fragment)  # --- Przykład Dziedziczenia w testach ---  class BaseTestUtility(unittest.TestCase):     """     Klasa bazowa dla testów pomocniczych, demonstrująca dziedziczenie.     """      def setUp(self):         """Metoda setUp w klasie bazowej.""" </pre>		2	2


		<pre> super().setUp() # Wywołuje setUp z unittest.TestCase self.base_value = 10 # print(f"\n[BaseTestUtility] setUp for {self._testMethodName}: base_value = {self.base_value}")  def tearDown(self):     """Metoda tearDown w klasie bazowej."""     # print(f"[BaseTestUtility] tearDown for {self._testMethodName}: Cleaning up base_value")     self.base_value = None # Resetowanie wartości     super().tearDown() # Wywołuje tearDown z unittest.TestCase  def test_base_feature(self):     """Test bazowej funkcjonalności."""     # print(f"[BaseTestUtility] Running test_base_feature with base_value = {self.base_value}")     self.assertEqual(self.base_value, 10)  class DerivedTestUtility(BaseTestUtility):     """     Klasa pochodna dziedzicząca z BaseTestUtility,     rozszerzająca funkcjonalność.     """     def setUp(self):         """Metoda setUp w klasie pochodnej, rozszerzająca         bazową."""         super().setUp() # Wywołuje setUp z BaseTestUtility         self.derived_value = 20         self.total_value = self.base_value + self.derived_value         # print(f"[DerivedTestUtility] setUp for {self._testMethodName}: derived_value = {self.derived_value}, total_value = {self.total_value}")      def tearDown(self):         """Metoda tearDown w klasie pochodnej."""         # print(f"[DerivedTestUtility] tearDown for {self._testMethodName}: Cleaning up derived_value") </pre>			
--	--	--	--	--	--

			<pre> self.derived_value = None self.total_value = None super().tearDown() # Wywołuje tearDown z BaseTestUtility  def test_derived_feature(self):     """Test funkcjonalności w klasie pochodnej."""     # print(f"[DerivedTestUtility] Running test_derived_feature with total_value = {self.total_value}")     self.assertEqual(self.total_value, 30) # 10 (z bazowej) + 20 (z pochodnej)  def test_access_base_feature_from_derived(self):     """Test dostępu do funkcjonalności bazowej z klasy pochodnej."""     # print(f"[DerivedTestUtility] Running test_access_base_feature_from_derived. base_value = {self.base_value}")     self.assertEqual(self.base_value, 10) # Dostęp do odziedziczonego atrybutu </pre>			
10	Programowanie funkcyjne	map	<pre> # quiz_project/quiz_player/player.py (fragment z metody QuizPlayer.play_quiz)  # ... (kod wcześniejszy - analiza wyników quizu, gdzie user_answers zawiera szczegóły każdej odpowiedzi) ...  # Filter dla niepoprawnych odpowiedzi incorrect_results = list(filter(lambda ans: not ans["is_correct"], user_answers))  # Użycie map do pobrania treści pytań, na które odpowiedziano błędnie # Funkcja 'map' przyjmuje funkcję (lambda) jako pierwszy argument incorrect_question_texts = list(map(lambda ans: ans["question_text"], incorrect_results))  if incorrect_question_texts: </pre>		1.5	1,5


		<pre>print("\nPytań, na które odpowiedziałeś/aś błędnie:")  for text in incorrect_question_texts:     print(f"- {text}")  # ... (kod późniejszy) ...</pre>			
	filter	<pre># quiz_project/quiz_player/player.py (fragment z metody QuizPlayer.play_quiz)  # ... (kod wcześniejszy - user_answers zawiera szczegóły każdej odpowiedzi) ...  # --- Analiza wyników z użyciem programowania funkcyjnego ---  # Filter dla poprawnych odpowiedzi # Funkcja 'filter' przyjmuje funkcję (lambda) jako pierwszy argument correct_results = list(filter(lambda ans: ans["is_correct"], user_answers)) num_correct = len(correct_results)  # Filter dla niepoprawnych odpowiedzi # Funkcja 'filter' przyjmuje funkcję (lambda) jako pierwszy argument incorrect_results = list(filter(lambda ans: not ans["is_correct"], user_answers)) num_incorrect = len(incorrect_results)  # ... (użycie map do pobrania treści pytań, na które odpowiedziano błędnie) ...</pre>		1.5	1, 5
	lambda	<pre># quiz_project/quiz_player/player.py (fragment z metody QuizPlayer.play_quiz)  # ... (kod wcześniejszy - user_answers zawiera szczegóły każdej odpowiedzi) ...  # --- Analiza wyników z użyciem programowania funkcyjnego ---  # Filter dla poprawnych odpowiedzi</pre>		1.5	1, 5

		<pre> # Funkcja lambda 'lambda ans: ans["is_correct"]' jest używana jako funkcja testująca correct_results = list(filter(lambda ans: ans["is_correct"], user_answers)) num_correct = len(correct_results)  # Filter dla niepoprawnych odpowiedzi # Funkcja lambda 'lambda ans: not ans["is_correct"]' jest używana jako funkcja testująca incorrect_results = list(filter(lambda ans: not ans["is_correct"], user_answers)) num_incorrect = len(incorrect_results)  # Użycie map do pobrania treści pytań, na które odpowiedziano błędnie # Funkcja lambda 'lambda ans: ans["question_text"]' jest używana jako funkcja transformująca incorrect_question_texts = list(map(lambda ans: ans["question_text"], incorrect_results)) if incorrect_question_texts:     print("\nPytań, na które odpowiedziałeś/aś błędnie:")     for text in incorrect_question_texts:         print(f"- {text}")  # ... (kod późniejszy) ... </pre>			
	reduce	<pre> def sum_list_elements(numbers: list[int]) -&gt; int:     """     Uses functools.reduce to sum all elements in a list of     numbers.     This function demonstrates the use of reduce.      Args:         numbers (list[int]): A list of integers.      Returns:         int: The sum of all elements.     """     if not isinstance(numbers, list) or not all(isinstance(n, int) for n in numbers): </pre>		1.5	1, 5


			<pre> raise TypeError("Input must be a list of integers.") return reduce(lambda acc, x: acc + x, numbers, 0) </pre>			
1 1	Wizualizacja danych	Wygenerowano wykres (np. matplotlib, seaborn)	<pre> # quiz_project/quiz_player/player.py (fragment z metody QuizPlayer.generate_and_save_results_chart)  import matplotlib.pyplot as plt # ... (inne importy i kod) ...  class QuizPlayer:     # ... (inne metody) ...      @staticmethod     def generate_and_save_results_chart(correct_count: int, incorrect_count: int, quiz_title: str):         """         Generates a pie chart showing the distribution of correct vs. incorrect answers.         """         labels: tuple[str, str] = ('Poprawne', 'Niepoprawne')         sizes = [correct_count, incorrect_count]         colors: tuple[str, str] = ('#4CAF50', '#F44336') # Green for correct, Red for incorrect         explode = (0.1, 0) # explode the 1st slice (Correct)          # Tworzenie figury i osi wykresu         fig1, ax1 = plt.subplots()          # Generowanie wykresu kołowego         ax1.pie(sizes, explode=explode, labels=labels, colors=colors,                 autopct='%1.1f%%', shadow=True, startangle=90, textprops={'fontsize': 12, 'color': 'white'})          # Ustawienie równych proporcji osi, aby koło było okrągłe         ax1.axis('equal')          # Ustawienie tytułu wykresu         plt.title(f'Wyniki quizu: {quiz_title}', fontsize=16, color='black') </pre>		2	

			# ... (kod zapisywania wykresu) ...			
	Zapisano wykres do pliku graficznego (.png lub .jpg)		<pre># quiz_project/quiz_player/player.py (fragment z metody QuizPlayer.generate_and_save_results_chart)  import matplotlib.pyplot as plt import os # ... (inne importy i kod) ...  class QuizPlayer:     # ... (inne metody) ...      @staticmethod     def generate_and_save_results_chart(correct_count: int, incorrect_count: int, quiz_title: str):         """         Generates a pie chart showing the distribution of correct vs. incorrect answers         and saves it to the 'reports' directory.         """         # ... (kod generujący wykres matplotlib) ...          # Katalog docelowy dla raportów         reports_dir = "reports"         if not os.path.exists(reports_dir):             os.makedirs(reports_dir)             print(f"Created directory: {reports_dir}")          # Konstruowanie nazwy pliku i pełnej ścieżki         chart_filename = f"wyniki_{quiz_title.replace(' ', '_').lower()}.png"         chart_filepath = os.path.join(reports_dir, chart_filename)          try:             # Zapisywanie wykresu do pliku graficznego (PNG)             plt.savefig(chart_filepath, bbox_inches='tight', dpi=100)</pre>		1.5	1, 5




			<pre> print(f"Wykres wyników został zapisany w: {chart_filepath}") except Exception as e:     print(f"Błąd podczas zapisywania wykresu: {e}") finally:     plt.close(fig1) # Zamykanie figury matplotlib po zapisie </pre>			
T 1 2	Testowa nie	Testy jednostkow e (assert, unittest, pytest)	<pre> # quiz_project/tests/test_models.py (fragmenty z klasy TestQuestion i TestQuiz)  import unittest # ... (inne importy) ...  class TestQuestion(unittest.TestCase):     """     Unit tests for the Question class.     Covers initialization, validation, display, correctness     check,     and serialization/deserialization.     """      def test_question_initialization_valid(self):         """Test Question initialization with valid data."""         question = Question("What is 2+2?", ["3", "4", "5"], 1)         # self.assertEqual: Sprawdza, czy dwie wartości są równe.         self.assertEqual(question.question_text, "What is 2+2?")         self.assertEqual(question.options, ["3", "4", "5"])         self.assertEqual(question.correct_answer_index, 1)      def test_question_initialization_empty_text_raises_error(self):         """Test Question initialization with empty question text."""         # self.assertRaisesRegex: Sprawdza, czy dany kod rzuca wyjątek z określonym komunikatem.         with self.assertRaisesRegex(ValueError, "Question text cannot be empty."):             Question("", ["a", "b"], 0) </pre>		1.5	1, 5


			<pre> def test_question_is_correct(self):     """Test the is_correct method for correct and     incorrect answers."""     question = Question("What is 2+2?", ["3", "4", "5"],     1)     # self.assertTrue / self.assertFalse: Sprawdza, czy     warunek jest prawdziwy/fałszywy.     self.assertTrue(question.is_correct(1)) # Oczekujemy     True     self.assertFalse(question.is_correct(0)) #     Oczekujemy False  def test_question_to_dict_from_dict(self):     """Test conversion to dictionary and back to     object."""     original_question = Question("Fav color?", ["Red",     "Blue"], 0)     q_dict = original_question.to_dict()      # self.assertIsInstance: Sprawdza, czy obiekt jest     instancją danej klasy/typu.     self.assertIsInstance(q_dict, dict)     self.assertEqual(q_dict["question_text"], "Fav     color?")      reconstructed_question =     Question.from_dict(q_dict)     self.assertIsInstance(reconstructed_question,     Question)      self.assertEqual(reconstructed_question.question_text,     original_question.question_text)  class TestQuiz(unittest.TestCase):     """Unit tests for the Quiz class."""     # ... (kod setUp) ...  def test_add_question(self):     """Test adding questions to the quiz.""" </pre>			
--	--	--	---	--	--	--

		<pre> quiz = Quiz("Test Add") quiz.add_question(self.q1) self.assertEqual(len(quiz.questions), 1) self.assertEqual(quiz.questions[0].question_text, "Q1 text")  def test_remove_question_invalid_index_raises_error(self):     """Test removing question with out-of-bounds index."""     quiz = Quiz("Test Remove Invalid", questions=[self.q1])     with self.assertRaisesRegex(IndexError, "Question index is out of bounds."):         quiz.remove_question(1)  # ... (pozostałe klasy testowe, np. dziedziczenie w testach) ... </pre>			
	Testy funkcjonalne	<pre> # quiz_project/tests/test_interactive_modules.py (fragmenty z klas TestQuizCreator i TestQuizPlayer)  import unittest from unittest.mock import patch from io import StringIO # ... (inne importy) ...  class TestQuizCreator(unittest.TestCase):     """     Unit tests for the QuizCreator class, focusing on its interactive functionality.     These tests can also be considered functional tests as they simulate user input     and verify the overall outcome of user-driven processes (e.g., quiz creation).     """     # ... (setUp and tearDown methods) ...      @patch('builtins.input', side_effect=['Test Quiz Title', 'Test Description', 'Q1 text?', 'Option A', 'Option B', '', '1', # Question 1 </pre>		1.5	1, 5

		<pre> ", # End adding questions 'test_quiz_output']) #  Filename to save def test_create_new_quiz_success(self, mock_input):     """     Test successful creation of a new quiz with valid     input.     This is a functional test because it simulates a     complete user flow     (providing all inputs for quiz creation) and verifies     the final state     (quiz saved, correct messages displayed).     """     QuizCreator.create_new_quiz()     output = self.held_output.getvalue()     self.assertIn("Quiz 'Test Quiz Title' został     utworzony. Teraz dodaj pytania.", output)     self.assertIn("Quiz został pomyślnie zapisany!",     output)     self.mock_save_quiz.assert_called_once() # Checks     the ultimate outcome  class TestQuizPlayer(unittest.TestCase):     """     Unit tests for the QuizPlayer class, focusing on its     interactive functionality.     These also serve as functional tests, simulating quiz     playback from start to end.     """     # ... (setUp and tearDown methods) ...      @patch('builtins.input', side_effect=['1', # Select the     only available quiz     '1', # Answer to Q1 (correct)     '2']) # Answer to Q2 (correct)     def test_play_quiz_success_all_correct(self,     mock_input):         """         Test successful playing of a quiz with all correct         answers. </pre>		
--	--	---	--	--

	<p>This is a functional test because it simulates the user playing a quiz from selection through answering all questions, and verifies the final results and side effects (chart generation).</p> <pre> """ QuizPlayer.play_quiz() output = self.held_output.getvalue() self.assertIn("Twój wynik: 2/2 poprawnych odpowiedzi.", output) self.assertIn("Szczegółowe wyniki zostały zapisane w raporcie graficznym.", output) self.mock_chart_gen.assert_called_once_with(2, 0, "Player Test Quiz") # Checks side effect </pre>			
Testy Integracyjne	<pre> # quiz_project/tests/test_interactive_modules.py (fragment z klasy TestQuizCreator)  import unittest from unittest.mock import patch from io import StringIO # ... (inne importy) ...  class TestQuizCreator(unittest.TestCase):     # ... (setUp and tearDown methods) ...      def _create_sample_quiz_file_for_editing(self):         """Helper to create a sample quiz object for editing tests."""         sample_q1 = Question("Original Q1", ["Opt1_A", "Opt1_B"], 0)         sample_q2 = Question("Original Q2", ["Opt2_C", "Opt2_D"], 1)         sample_quiz = Quiz("Original Test Quiz", "Original description", questions=[sample_q1, sample_q2])         # W tym miejscu faktycznie zapisujemy quiz (za pomocą QuizDataManager)         # do tymczasowego katalogu, co jest częścią integracji dla setupu testowego.         QuizDataManager.save_quiz(sample_quiz, "original_test_quiz", self.test_dir)         return sample_quiz </pre>		1.5	1, 5


	<pre> @patch('quiz_data.manager.QuizDataManager.list_available_quizzes', return_value=["original_test_quiz"])  @patch('quiz_data.manager.QuizDataManager.load_quiz')     @patch('builtins.input', side_effect=['1', # Wybierz "original_test_quiz" do edycji                                 '1', # Edytuj tytuł/opis                                 'New Title', # Nowy tytuł                                 'New Description', # Nowy opis                                 '5', # Opcja 5: Zapisz i wyjdź                                 'original_test_quiz', # Nazwa pliku do zapisu (nadpisz)                                 'tak']) # Potwierdź nadpisanie     def test_edit_quiz_title_and_description(self, mock_input, mock_load, mock_list):         """         Testuje edycję tytułu i opisu istniejącego quizu.         To jest test integracyjny, ponieważ obejmuje interakcję między:         - QuizCreator (logika edycji UI)         - QuizDataManager (ładowanie i zapis quizu)         - Klasami modelu Quiz i Question         """         # Ustaw mock load_quiz, aby zwracał nasz przykładowy quiz         sample_quiz = self._create_sample_quiz_file_for_editing()         mock_load.return_value = sample_quiz          QuizCreator.edit_existing_quiz() # Wywołanie funkcji, która integruje wiele komponentów          output = self.held_output.getvalue()         self.assertIn("Quiz 'Original Test Quiz' został wczytany do edycji.", output)         self.assertIn("Tytuł i opis zaktualizowane.", output)         self.assertIn("Zakończono edycję quizu.", output) </pre>			
--	---	--	--	--



	<pre> # Weryfikacja, czy save_quiz zostało wywołane ze zmodyfikowanym quizem self.mock_save_quiz.assert_called_once() saved_quiz = self.mock_save_quiz.call_args[0][0] # Pobierz obiekt Quiz przekazany do zapisu self.assertEqual(saved_quiz.title, "New Title") self.assertEqual(saved_quiz.description, "New Description") self.assertEqual(len(saved_quiz.questions), 2) # Pytania powinny pozostać niezmienione </pre>			
Testy graniczne / błędne dane	<pre> # quiz_project/tests/test_models.py (fragmenty z klasy TestQuestion i TestQuiz)  import unittest # ... (inne importy) ...  class TestQuestion(unittest.TestCase):     """     Unit tests for the Question class.     Covers initialization, validation, display, correctness     check,     and serialization/deserialization.     """      def test_question_initialization_empty_text_raises_error(self):     """     Test Question initialization with empty question     text.     Test graniczny: pusta treść pytania.     """     with self.assertRaisesRegex(ValueError, "Question text cannot be empty."):         Question("", ["a", "b"], 0)     with self.assertRaisesRegex(ValueError, "Question text cannot be empty."):         Question(" ", ["a", "b"], 0) </pre>		1.5	1, 5


	<pre> def test_question_initialization_empty_options_raises_error(s elf):     """     Test Question initialization with empty options list.     Test graniczny: pusta lista opcji.     """     with self.assertRaisesRegex(ValueError, "Options must be a non-empty list."):         Question("Test?", [], 0)  def test_question_initialization_invalid_option_type_raises_er ror(self):     """     Test Question initialization with non-string options.     Test błędnych danych: opcje nie są stringami lub są puste.     """     with self.assertRaisesRegex(ValueError, "All options must be non-empty strings."):         Question("Test?", ["a", 123], 0)     with self.assertRaisesRegex(ValueError, "All options must be non-empty strings."):         Question("Test?", ["a", " "], 0) # Pusty string jako opcja  def test_question_initialization_invalid_index_raises_error(self ):     """     Test Question initialization with an out-of-bounds or invalid index.     Test graniczny/błędnych danych: indeks poza zakresem lub nie jest liczbą.     """     with self.assertRaisesRegex(ValueError, "Correct answer index is out of bounds or not an integer."):         Question("Test?", ["a", "b"], 2) # Indeks poza zakresem </pre>			
--	---	--	--	--

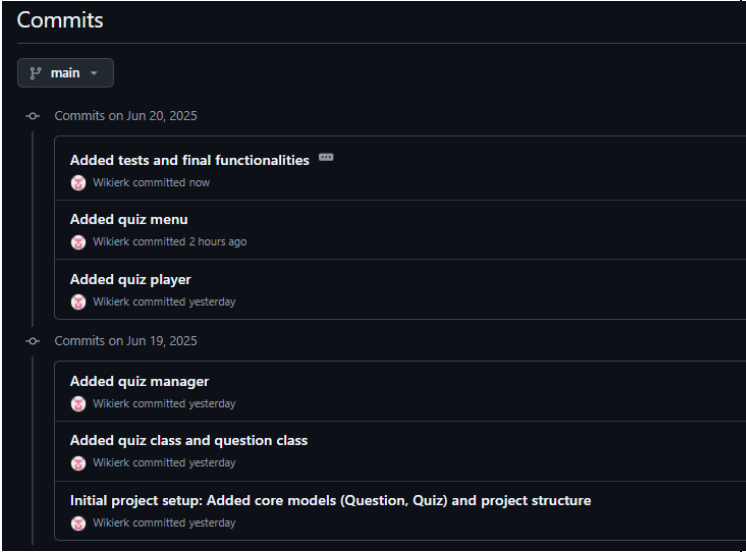




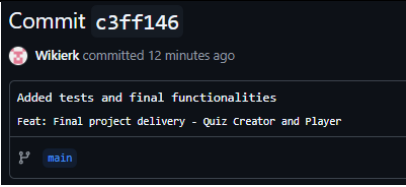


	<pre>         with self.assertRaisesRegex(ValueError, "Correct         answer index is out of bounds or not an integer."):             Question("Test?", ["a", "b"], -1) # Indeks ujemny         with self.assertRaisesRegex(ValueError, "Correct         answer index is out of bounds or not an integer."):             Question("Test?", ["a", "b"], "0") # Nieprawidłowy         typ indeksu  class TestQuiz(unittest.TestCase):     """     Unit tests for the Quiz class.     """     # ... (kod setUp) ...      def test_quiz_initialization_empty_title_raises_error(self):     """     Test Quiz initialization with empty title.     Test graniczny: pusty tytuł quizu.     """     with self.assertRaisesRegex(ValueError, "Quiz title cannot be empty."):         Quiz("")     with self.assertRaisesRegex(ValueError, "Quiz title cannot be empty."):         Quiz(" ")      def test_remove_question_invalid_index_raises_error(self):     """     Test removing question with out-of-bounds index.     Test graniczny/błędnych danych: indeks poza zakresem.     """     quiz = Quiz("Test Remove Invalid", questions=[self.q1])     with self.assertRaisesRegex(IndexError, "Question index is out of bounds."):         quiz.remove_question(1) # Indeks za wysoki </pre>			
--	--	--	--	--

	<pre> with self.assertRaisesRegex(IndexError, "Question index is out of bounds."):     quiz.remove_question(-1) # Indeks ujemny  def test_quiz_from_dict_missing_keys_raises_error(self):     """     Test from_dict with missing required keys.     Test błędnych danych: uszkodzone dane JSON     (brak kluczy).     """     with self.assertRaises(KeyError):         Quiz.from_dict({"description": "incomplete"}) #         Brak klucza 'title'      # Test z brakującymi kluczami dla pytania     with self.assertRaises(KeyError):         Quiz.from_dict({             "title": "Valid Quiz",             "questions": [                 {"question_text": "Q1", "options": ["A", "B"]}, # Brak 'correct_answer_index'             ]         }) </pre>			
Testy wydajności (np. czas wykonania, timeit)	<pre> def measure_function_performance(func_name: str, setup_code: str, num_runs: int = 100000) -&gt; float:     """     Measures the execution time of a given function     using timeit.     This demonstrates performance testing using timeit.      Args:         func_name (str): The name of the function to         measure (as a string).         setup_code (str): Setup code for the timeit         environment (e.g., imports).         num_runs (int): Number of times to execute the         function for measurement.      Returns:         float: The average execution time in seconds. </pre>		1.5	1, 5

		<pre> """ # timeit.timeit wykonuje dany kod 'number' razy # setup to kod, który jest wykonywany raz przed pomiarom time_taken = timeit.timeit(stmt=f"{func_name}()", setup=setup_code, number=num_runs) average_time = time_taken / num_runs print(f"[{func_name}] Średni czas wykonania ({num_runs} przebiegów): {average_time:.6f} sekund") return average_time </pre>			
	Testy pamięci memory_profiler	<pre> @profile # Dekorator z memory_profiler do mierzenia zużycia pamięci def generate_large_data(size_mb: int):     """     Generates a large list to demonstrate memory usage.     This function is used for memory profiling with     memory_profiler.      Args:         size_mb (int): Desired size of the data in         megabytes.     """     print(f"\n[memory_profiler] Generowanie danych o     rozmiarze {size_mb} MB...")     num_elements = int(size_mb * 1024 * 1024 / 28)     data = [i for i in range(num_elements)]     print(f"[memory_profiler] Wygenerowano listę z     {len(data)} elementami.") </pre>		1.5	1, 5
	Test jakości kodu (flake8, pylint)	<pre> # --- Funkcja do demonstracji testu jakości kodu (Pylint) --- def run_pylint_check(file_path: str) -&gt; int:     """     Demonstrates how to programmatically run a Pylint     check on a given file.     This fulfills the requirement for code quality testing.      Args:         file_path (str): The path to the Python file to check.      Returns: </pre>		1.5	1, 5


			<pre> int: The exit code of Pylint (0 if no errors/warnings, &gt;0 otherwise). """  print(f"\n[Code Quality Check] Uruchamiam Pylint dla pliku: {file_path}")  # Ścieżka do skryptu pylint - często jest w PATH, ale można podać pełną ścieżkę # pylint_executable = "pylint"  # Argumenty dla pylint: # --rcfile=none - ignoruje globalne pliki konfiguracyjne (dla czystego przykładu) # --disable=all - wyłącza wszystkie wiadomości, a następnie włącza konkretne (przykładowo) # --enable=E,W,C - włącza tylko błędy (Error), ostrzeżenia (Warning) i konwencje (Convention) # file_path - plik do sprawdzenia pylint_args = [file_path, "--reports=no", "-- disable=C0114,C0115,C0116"] # Przykładowe wyłączenie brakujących docstringów  # Wywołanie pylint za pomocą os.system lub pylint.lint.run_pylint # Używamy zaimplementowanego obiektu pylint_runner do uruchomienia exit_code = pylint_runner.run_pylint(pylint_args)  if exit_code == 0:     print(f"[Code Quality Check] Pylint zakończył się sukcesem dla {file_path}. Kod spełnia standardy jakości.") else:     print(f"[Code Quality Check] Pylint znalazł problemy w {file_path}. Kod wyjścia: {exit_code}")  return exit_code </pre>			
13	Wersjonowanie	Repozytorium GIT	<a href="https://github.com/Wikierk/quiz-project.git">https://github.com/Wikierk/quiz-project.git</a>		1	1

		Historia commitów			1	1
--	--	-------------------	--	---	---	---

Obszar	Wymagania	Kod		Przyznane pkt	Pkt max
	Lin do GitHub	<a href="https://github.com/Wikierk/quiz-project.git">https://github.com/Wikierk/quiz-project.git</a>		1	1
	Opis commitów			1	1
Dokumentacja	Plik README.md(cel,autorzy, uruchamianie)	<p>Cel: Aplikacja do tworzenia, edytowania, uruchamiania quizów</p> <p>Autorzy: Wiktor Pacak, Wojciech Opara</p> <p>Instrukcja uruchomienia</p> <p>1. Sklonuj lub pobierz projekt:</p> <pre>git clone https://github.com/Wikierk/quiz-project.git cd nazwa_repozytorium # Przejdź do katalogu głównego projektu (np. quiz_project)</pre> <p>2. Zainstaluj wymagane zależności:</p> <pre>pip install -r requirements.txt</pre>		1	1

		3. Uruchom aplikację: python main.py			
	Przykładowe dane wejściowe i wyjściowe	<p>--- Rozpoczęcie tworzenia nowego quizu</p> <p>--- Podaj tytuł quizu (np. 'Geografia Polski'): Podstawy Pythona</p> <p>Podaj krótki opis quizu (opcjonalnie): Quiz sprawdzający podstawową wiedzę o Pythonie.</p> <p>Quiz 'Podstawy Pythona' został utworzony.</p> <p>Teraz dodaj pytania.</p> <p>--- Dodawanie nowego pytania ---</p> <p>Wpisz treść pytania (naciśnij Enter, aby zakończyć dodawanie pytań): Jaki jest wynik 2 + 2?</p> <p>Wpisuj opcje odpowiedzi. Naciśnij Enter na pustej linii, aby zakończyć dodawanie opcji.</p> <p>Opcja 1: 3</p> <p>Opcja 2: 4</p> <p>Opcja 3: 5</p> <p>Opcja 4: Dostępne opcje:</p> <ol style="list-style-type: none"> <li>1. 3</li> <li>2. 4</li> <li>3. 5</li> </ol> <p>Wpisz numer poprawnej odpowiedzi: 2</p> <p>Pytanie dodane pomyślnie!</p> <p>--- Dodawanie nowego pytania --- Wpisz treść pytania (naciśnij Enter, aby zakończyć dodawanie pytań):</p> <p>Zakończono dodawanie pytań.</p> <p>Podaj nazwę pliku, pod którą zapisać quiz (bez rozszerzenia .json): python_basics_quiz</p>	<input type="checkbox"/>	1.5	1.5

			<p>Quiz został pomyślnie zapisany!</p> <p>Naciśnij Enter, aby kontynuować...</p> <p>#### Odtwarzanie quizu:</p> <p>--- Rozpoczęcie odtwarzania quizu ---</p> <p>Dostępne quizy:</p> <p>1. python_basics_quiz</p> <p>Wybierz numer quizu do odtworzenia: 1</p> <p>Quiz 'Podstawy Pythona' loaded successfully from data\quiz_examples\python_basics_quiz.json</p> <p>--- Rozpoczęcie quizu: Podstawy Pythona ---</p> <p>Opis: Quiz sprawdzający podstawową wiedzę o Pythonie.</p> <p>--- Pytanie 1/1 --- Pytanie: Jaki jest wynik 2 + 2?</p> <p>1. 3 2. 4 3. 5</p> <p>Wpisz numer odpowiedzi: 2</p> <p>Poprawna odpowiedź!</p> <p>--- Koniec quizu! --- Twój wynik: 1/1 poprawnych odpowiedzi.</p> <p>Szczegółowe wyniki zostały zapisane w raporcie graficznym. Wykres wyników został zapisany w: reports\wyniki_podstawy_pythona_quiz.png</p> <p>Naciśnij Enter, aby kontynuować...</p> <p>(Wykres zostanie wygenerowany w katalogu `reports/`)</p>			
--	--	--	--	--	--	--

	Diagram klas lub struktura modułów	<pre> quiz_project/ ├── main.py          # Główny plik uruchamiający aplikację ├── models/          # Pakiet z definicjami klas modelu │   ├── init.py     # Inicjalizuje pakiet 'models' │   ├── question.py # Definiuje klasę Question │   └── quiz.py      # Definiuje klasę Quiz ├── quiz_data/       # Pakiet do zarządzania danymi quizów │   ├── init.py     # Inicjalizuje pakiet 'quiz_data' │   └── manager.py  # Klasa QuizDataManager do zapisu/odczytu JSON ├── quiz_creator/    # Pakiet do tworzenia i edycji quizów │   ├── init.py     # Inicjalizuje pakiet 'quiz_creator' │   └── creator.py  # Klasa QuizCreator z logiką tworzenia/edycji UI ├── quiz_player/     # Pakiet do odtwarzania quizów │   ├── init.py     # Inicjalizuje pakiet 'quiz_player' │   └── player.py   # Klasa QuizPlayer z logiką rozgrywki i wizualizacji ├── utils/           # Pakiet z ogólnymi funkcjami pomocniczymi │   ├── init.py     # Inicjalizuje pakiet 'utils' │   └── helpers.py  # Zawiera funkcje rekurencyjne, dekoratory, testy wydajności/pamięci ├── data/            # Katalog na przykładowe pliki JSON z quizami │   └── quiz_examples/ │       └── example_quiz.json # Przykładowy plik quizu (tworzony przez aplikację) ├── reports/         # Katalog na wygenerowane raporty graficzne (wykresy) ├── tests/           # Pakiet z testami jednostkowymi i integracyjnymi │   ├── init.py     # Inicjalizuje pakiet 'tests' │   ├── test_models.py # Testy dla Question i Quiz (w tym dziedziczenie w testach) │   ├── test_data_manager.py # Testy dla QuizDataManager │   ├── test_interactive_modules.py # Testy dla QuizCreator i QuizPlayer (z mockowaniem) └── requirements.txt # Lista zależności projektu </pre>		2	2
		Suma		100	100