Nr	Obszar	Wymaganie	KOD		Przyznane pkt	Pkt max
1	UI	JEST	-	☐		

		Wprowadzanie danych	def add_pokoj(self):
			"""Dodanie nowego pomieszczenia."""
			try:
			numer = int(input("Numer pokoju: "))
			czynsz = float(input("Czynsz (zł): "))...
				☑		2

		Wyświetlanie danych	def show_pokoje(self):
			"""Wyświetlanie wszystkich pomieszczeń."""
			if not self.pokoje:
			print("Brak pokoi w systemie.")
			for pokoj in self.pokoje:
			lokalizacja_str = f"{pokoj.lokalizacja['miasto']}, {pokoj.lokalizacja['ulica']} {pokoj.lokalizacja['numer_domu']}"
			print(f"Pokój {pokoj.numer}, Czynsz: {pokoj.czynsz} zł, Lokalizacja: {lokalizacja_str}")
				☑		2

		Zmiana danych	def edit_pokoj(self):
			…
			czynsz = input(f"Nowy czynsz (obecny: {pokoj.czynsz} zł, Enter aby pominąć): ")
			if czynsz:
			pokoj.czynsz = float(czynsz)
			…
				☑		2

		Wyszukiwanie danych	def filter_czynsz(self): """Filtrowanie pokoi według czynszu przy użyciu lambda i filtra.""" … limit = float(input("Podaj maksymalny czynsz: ")) znalezione = list(filter(lambda p: p.czynsz <= limit, self.pokoje)) …	☑		2
		Przedstawienie wyników	def plot_rent_distribution(pokoje): """Tworzenie histogramu rozkładu czynszów.""" rents = [p.czynsz for p in pokoje] plt.hist(rents, bins=10, edgecolor="black") …	☑		2

2	Podstawy	Zmienne	# Globalna zmienna dla nazwy pliku danych
			DATA_FILE = "data.json"
				☑		2

		typy danych	class Pokoj: """Podstawowa klasa dla pokoju."""	☑		2

			def __init__(self, numer, czynsz, najemca=None, lokalizacja=None): … self.lokalizacja = lokalizacja or {"miasto": "", "ulica": "", "numer_domu": ""} # Słownik dla lokalizacji			
		komentarze	def to_dict(self): """Konwersja obiektu pokoju do słownika.""" ….	☑		1

		operatory	def add_pokoj(self):
			"""Dodanie nowego pomieszczenia."""
			try:
			…
			assert czynsz > 0, "Czynsz musi być dodatni!"
				☑		1,5

		Instrukcje warunkowe (if, elif, else)	def menu(self):
			"""Menu główne z wyborem operacji."""
			while True:
			…
			if choice == "1":
			self.show_pokoje()
			elif choice == "2":
			self.filter_czynsz()
			…
				☑		3

		Instrukcje iteracyjne				

		for	def add_pokoj(self):
			"""Dodanie nowego pomieszczenia."""
			…
			# Sprawdzanie, czy pokój z takim numerem i lokalizacją już istnieje
			for pokoj in self.pokoje:
			…
				☑		2

		while	def menu(self):
			"""Menu główne z wyborem operacji."""
			while True:
			…
				☑		2

		Operacje wejścia (input)	def menu(self):
			"""Menu główne z wyborem operacji."""
			while True:
			…
			choice = input("> ")
			…
				☑		1,5

		Operacje wyjścia (print)	def add_pokoj(self): … print("Pokój dodany.") …	☑		1,5

		Funkcje z parametrami i wartościami zwracanymi	def count_free_rooms_recursive(self, pokoje, index=0):
			"""Rekursywne zliczanie dostępnych pokoi."""
			if index >= len(pokoje):
				☑		2

			return 0 return (1 if pokoje[index].najemca is None else 0) + self.count_free_rooms_recursive(pokoje, index + 1)			

		Funkcje rekurencyjne	def count_free_rooms_recursive(self, pokoje, index=0):
			"""Rekursywne zliczanie dostępnych pokoi."""
			if index >= len(pokoje):
			return 0
			return (1 if pokoje[index].najemca is None else 0) + self.count_free_rooms_recursive(pokoje, index + 1)
				☑		3

		Funkcje przyjmujące inne funkcje jako argumenty	def apply_function_to_rents(self, func): """Zastosowanie przeniesionej funkcji do płatności leasingowych.""" return reduce(lambda acc, p: acc + func(p.czynsz), self.pokoje, 0)	☑		3

		Dekoratory	def log_execution_time(func):
			"""Dekoder do rejestrowania czasu wykonania metody."""
			def wrapper(*args, **kwargs):
			start_time = time.time()
			result = func(*args, **kwargs)
			print(f"Metoda {func.__name__} została wykonana w {time.time() - start_time:.2f} sek.")
			return result
			return wrapper
				☑		1,5

3	Kontenery	Użycie listy	class App:
			"""Główna klasa aplikacji do zarządzania wynajmem."""
			def __init__(self, filename):
			…
			self.pokoje = self.load_pokoje()
			…
				☑		2

		Użycie słownika	def add_pokoj(self):
			"""Dodanie nowego pomieszczenia."""
			try:
			…
			lokalizacja = {"miasto": miasto, "ulica": ulica, "numer_domu": numer_domu}
				☑		2

		Użycie zbioru	class App:	☑		1,5

			"""Główna klasa aplikacji do zarządzania wynajmem.""" def __init__(self, filename): … self.numbers_set = set(p.numer for p in self.pokoje) # Wiele numerów pokoi			
		Użycie krotki		☐		1,5

4	Przestrzenie nazw	Zastosowano zmienne lokalne	def add_pokoj(self):
			"""Dodanie nowego pomieszczenia."""
			try:
			numer = int(input("Numer pokoju: "))
			…
				☑		1,5

		Zastosowano zmienne globalne	# Globalna zmienna dla nazwy pliku danych
			DATA_FILE = "data.json"
				☑		1,5

		Zastosowano zakresy funkcji	def load_data(filename="data.json"): """Ładowanie danych z pliku JSON.""" …	☑		1,5
		Zastosowano zakresy klas	class Najemca: """Klasa reprezentująca najemca.""" …	☑		1,5

5	Moduły i pakiety	Projekt podzielony na moduły (import, _init_)	# Główny moduł do uruchamiania systemu zarządzania wynajmem
			from models import App
			
			# Globalna zmienna dla nazwy pliku danych
			DATA_FILE = "data.json"
			
			def main():
			"""Inicjalizacja i uruchomienie aplikacji."""
			app = App(DATA_FILE)
			app.run()
			
			if __name__ == "__main__":
			main()
				☑		2

Nr	Obszar	Wymaganie	KOD		Przyznane pkt	Pkt max

		Własne pakiety/funkcje pomocnicze w osobnych plikach .py	# Funkcje pomocnicze do pracy z plikami
			import json
			
			def load_data(filename="data.json"):
			  """Ładowanie danych z pliku JSON."""
			  try:
			  with open(filename, "r") as f:
			  return json.load(f)
			  except FileNotFoundError:
			  print(f"Plik {filename} nie istnieje, tworzenie nowego.")
			  return {}
			  except json.JSONDecodeError:
			  print("Błąd wczytywania danych JSON.")
			  return {}
			  finally:
			  print("Zakończono próbę wczytania danych.")
			
			def save_data(data, filename="data.json"):
			  """Zapisywanie danych do pliku JSON."""
			  try:
			  with open(filename, "w") as f:
			  json.dump(data, f, indent=4)
			  except IOError as e:
			  print(f"Błąd zapisu: {e}")
			  finally:
			  print("Zakończono próbę zapisu danych.")
				☑		2

6	Obsługa błędów	Obsługa wyjątków (try, except, finally)	def load_data(filename="data.json"):
			  """Ładowanie danych z pliku JSON."""
			  try:
			  with open(filename, "r") as f:
			  return json.load(f)
			  except FileNotFoundError:
			  print(f"Plik {filename} nie istnieje, tworzenie nowego.")
			  return {}
			…
				☑		2

		Użycie assert do testów i walidacji	class Pokoj:
			  """Podstawowa klasa dla pokoju."""
			  def __init__(self, numer, czynsz, najemca=None, lokalizacja=None):
			  assert czynsz > 0, "Czynsz musi być dodatni!"
				☑		1,5

7	Łańcuchy znaków	Operacje na stringach (m.in. formatowanie, dzielenie, wyszukiwanie)	def add_pokoj(self): udogodnienia = input("Udogodnienia (oddzielone przecinkami): ").split(",")	☑		2

8	Obsługa plików	Odczyt z plików .txt, .csv, .json, .xml (min. 1)	def load_data(filename="data.json"):
			"""Ładowanie danych z pliku JSON."""
			try:
			with open(filename, "r") as f:
			return json.load(f)
			…
				☑		2

		Zapis do plików .txt, .csv, .json, .xml (min. 1)	def save_data(data, filename="data.json"):
			"""Zapisywanie danych do pliku JSON."""
			try:
			with open(filename, "w") as f:
			json.dump(data, f, indent=4)
			…
				☑		2

9	OOP	Klasy	class Pokoj: """Podstawowa klasa dla pokoju.""" …	☑		2

		Metody	class Pokoj:
			"""Podstawowa klasa dla pokoju."""
			
			def to_dict(self):
			"""Konwersja obiektu pokoju do słownika."""
			return {
			"numer": self.numer,
			"czynsz": self.czynsz,
			"najemca": self.najemca.to_dict() if self.najemca else None,
			"lokalizacja": self.lokalizacja
			}
				☑		2

		Konstruktory	class Pokoj:
			"""Podstawowa klasa dla pokoju."""
			def __init__(self, numer, czynsz, najemca=None, lokalizacja=None):
			assert czynsz > 0, "Czynsz musi być dodatni!"
			self.numer = numer
			self.czynsz = czynsz
			self.najemca = najemca
			self.lokalizacja = lokalizacja or {"miasto": "", "ulica": "", "numer_domu": ""} # Słownik dla lokalizacji
				☑		2

		Dziedziczenie	class PremiumPokoj(Pokoj):
			"""Ocena za pokoje premium z dodatkowymi udogodnieniami."""
			def __init__(self, numer, czynsz, najemca=None, lokalizacja=None, udogodnienia=None):
			super().__init__(numer, czynsz, najemca, lokalizacja)
			self.udogodnienia = udogodnienia or []
				☑		2

10	Programowanie funkcyjne	map	@log_execution_time def save(self, filename="data.json"): """Zapisywanie danych w formacie JSON.""" from utils import save_data pokoje_dict = list(map(lambda p: p.to_dict(), self.pokoje))	☑		1,5

		filter	def filter_czynsz(self):
			"""Filtrowanie pokoi według czynszu przy użyciu lambda i filtra."""
			try:
			limit = float(input("Podaj maksymalny czynsz: "))
			znalezione = list(filter(lambda p: p.czynsz <= limit, self.pokoje))
			…
				☑		1,5

		lambda	lambda p: p.czynsz <= limit	☑		1,5
		reduce	def apply_function_to_rents(self, func): """Zastosowanie przeniesionej funkcji do płatności leasingowych.""" return reduce(lambda acc, p: acc + func(p.czynsz), self.pokoje, 0)	☐		1,5
11	Wizualizacja danych	Wygenerowano wykres (np. matplotlib, seaborn)	plt.hist(rents, bins=10, edgecolor="black")	☑		2
		Zapisano wykres do pliku graficznego (.png lub .jpg)	plt.savefig("rent_distribution.png")	☑		1,5
T12	Testowanie	Testy jednostkowe (assert, unittest, pytest)	def test_najemca_to_dict(self): """Test zamiany najemcy na słownictwo.""" expected = {"imie": "Jan", "nazwisko": "Kowalski", "email": "jan@example.com"} self.assertEqual(self.najemca.to_dict(), expected)	☑		1,5
		Testy funkcjonalne	def test_functional_filter_czynsz(self): """Test funkcjonalny: filtrowanie według czynszu."""	☑		1,5

			filtered = list(filter(lambda p: p.czynsz <= 1200.0, self.app.pokoje)) self.assertEqual(len(filtered), 1) self.assertEqual(filtered[0].numer, 1)			

		Testy Integracyjne	def test_integration_save_load(self):
			"""Test integracji: zapisywanie i ładowanie danych."""
			try:
			
			self.app.save("test_data.json")
			
			self.assertTrue(os.path.exists("test_data.json"), "Plik test_data.json nie został utworzony")
			loaded_data = load_data("test_data.json")
			self.assertIn("pokoje", loaded_data, "Klucz 'pokoje' nie istnieje")
			
			self.assertEqual(len(loaded_data["pokoje"]), 2)
			
			self.assertEqual(loaded_data["pokoje"][0]["numer"], 1)
			finally:
			if os.path.exists("test_data.json"):
			
			os.remove("test_data.json")
				☑		1,5

		Testy graniczne / błędne dane	def test_invalid_czynsz(self): """Test przypadku granicznego: ujemny czynsz.""" with self.assertRaises(AssertionError): Pokoj(3, -1000.0)	☑		1,5
		Testy wydajności (np. czas wykonania, timeit)	def test_performance_save(self): """Test wydajności: czas przechowywania danych.""" execution_time = timeit.timeit(lambda: self.app.save("test_data.json"), number=100) self.assertLess(execution_time, 1.0, "Zapisywanie jest zbyt wolne")	☑		1,5
		Testy pamięci memory_profiler	@profile def test_memory_save(self): """Test pamięci: zużycie pamięci podczas zapisywania danych.""" self.app.save("test_data.json")	☑		1,5
		Test jakości kodu (flake8, pylint)		☐		1,5
13	Wersjonowanie	Repozytorium GIT	.idea Updated methods for adding and	☑		1

			editing apartments 4 days ago __pycache__ Updated methods for adding and editing apartments 4 days ago README.md Update README.md 4 days ago data.json Updated methods for adding and editing apartments 4 days ago main.py Dodano główne pliki projektu 2 weeks ago models.py Updated methods for adding and editing apartments 4 days ago test.py Updated methods for adding and editing apartments 4 days ago utils.py Dodano główne pliki projektu 2 weeks ago visualization.py Dodano główne pliki projektu 2 weeks ago			
		Historia commitów	$ git log --oneline 43bab45 (HEAD -> main, origin/main) Updated methods for adding and editing apartments 65cb875 Merge branch 'main' of https://github.com/Nuggetsik/WynajemMieszkan 334f3fa Dodano główne pliki projektu 0e713e4 Update README.md 9981e43 first commit	☑		1

Nr	Obszar	Wymaganie	KOD		Przyznane pkt	Pkt max
		Link do GitHub	https://github.com/Nuggetsik/WynajemMieszkan	☑		1
		Opis commitów	Update README.md Nuggetsik Nuggetsik authored 4 days ago Verified Updated methods for adding and editing apartments	☑		1

			Nuggetsik Nuggetsik committed 4 days ago			
14	Dokumentacja	Plik README.md (cel, autorzy, uruchamianie)	Wynajem Mieszkan Aplikacja do zarządzania wynajmem mieszkań napisana w języku Python. Umożliwia dodawanie, edytowanie, przeglądanie i filtrowanie pokoi (mieszkań) na podstawie czynszu. Projekt wykorzystuje programowanie obiektowe i funkcyjne, obsługę plików JSON oraz testy jednostkowe…	☑		1,5
		Przykładowe dane wejściowe i wyjściowe	Użycie Po uruchomieniu aplikacji (python main.py) wyświetli się menu tekstowe z opcjami: === System zarządzania wynajmem mieszkań === 1. Pokaż pokoje 2. Filtruj wg czynszu 3. Zapisz 4. Wyjście 5. Dodaj pokój 6. Edytuj pokój 7. Pokaż statystyki (rekurencja) 8. Pokaż wizualizację > Przykład: Dodawanie pokoju Wejście: > 5 Numer pokoju: 1 Czynsz (zł): 1200 Miasto: Gdańsk Ulica: Długa Numer domu: 20 Czy pokój jest premium? (t/n): t Czy pokój ma najemcę? (t/n): t Imię najemcy: Anna Nazwisko najemcy: Nowak Email najemcy: anna@example.com Udogodnienia (oddzielone przecinkami): WiFi,Balkon Wyjście: Pokój dodany.	☑		2
		Diagram klas lub struktura modułów	ProjectPython_wynajmem_mieszkan ├── main.py ├── models.py ├── test.py ├── utils.py └── visualization.py	☑		2

	SUMA		