

Projekt zaliczeniowy – Języki skryptowe (Python)

Tytuł projektu: System zarządzania wynajmem mieszkań

Autorzy: Mykyta Lytvyn, Danylo Rushchak

Grupa: Informatyka, rok II

Data oddania: 23.06.2025

1. Cel projektu

Celem projektu było stworzenie aplikacji w języku Python do zarządzania wynajmem mieszkań. Aplikacja umożliwia dodawanie, edytowanie, przeglądanie i filtrowanie pokoi (mieszkań) na podstawie czynszu. Projekt łączy programowanie obiektowe (OOP) i funkcyjne, z naciskiem na obsługę błędów, testowanie jednostkowe oraz modularną strukturę kodu.

2. Zakres funkcjonalny

Stworzenie systemu do zarządzania wynajmem mieszkań, który pozwala na:

- Dodawanie nowych pokoi z uwzględnieniem numeru, czynszu, lokalizacji i najemcy.
- Edytowanie istniejących pokoi (czynsz, najemca, udogodnienia dla pokoi premium).
- Filtrowanie pokoi według maksymalnego czynszu.
- Zliczanie wolnych pokoi za pomocą rekurencji.
- Wizualizację rozkładu czynszu z użyciem matplotlib.
- Zapis i odczyt danych w formacie JSON.

Funkcje aplikacji

- **Dodawanie pokoi:** Użytkownik może dodać pokój (zwykły lub premium) z numerem, czynszem, lokalizacją (miasto, ulica, numer domu), najemcą i udogodnieniami (dla pokoi premium).
- **Edytowanie pokoi:** Możliwość zmiany czynszu, najemcy lub udogodnień po identyfikacji pokoju przez numer i lokalizację.
- **Przeglądanie pokoi:** Wyświetlanie listy wszystkich pokoi z ich szczegółami (czynsz, lokalizacja, najemca, udogodnienia).
- **Filtrowanie pokoi:** Filtrowanie pokoi na podstawie maksymalnego czynszu z użyciem funkcji filter i lambda.
- **Zliczanie wolnych pokoi:** Rekursywne obliczanie liczby pokoi bez najemcy.
- **Wizualizacja danych:** Generowanie wykresu rozkładu czynszu.
- **Zapis danych:** Zapisywanie danych pokoi do pliku JSON i ich wczytywanie.

Zakres funkcjonalny (co zostało zaimplementowane)

- Klasy Najemca, Pokoj, PremiumPokoj i App do zarządzania danymi.
- Walidacja danych wejściowych (np. dodatni czynsz, unikalność numeru i lokalizacji).
- Obsługa błędów z użyciem try-except i assert.
- Testy jednostkowe, funkcjonalne, integracyjne i graniczne z użyciem unittest i memory_profiler.
- Funkcjonalne podejście z użyciem map, filter, reduce i lambda.
- Dekorator @log_execution_time do mierzenia czasu wykonania metod.
- Wczytywanie i zapisywanie danych w formacie JSON.

3. Struktura projektu

Opis plików i folderów

- **main.py**: Główny moduł uruchamiający aplikację, inicjalizuje klasę App i wywołuje metodę run.
- **models.py**: Zawiera klasy Najemca, Pokoj, PremiumPokoj i App, implementujące logikę zarządzania pokojami.
- **utils.py**: Moduł z funkcjami pomocniczymi do wczytywania (load_data) i zapisywania (save_data) danych w formacie JSON.
- **visualization.py**: Moduł z funkcją plot_rent_distribution do generowania wykresów rozkładu czynszu za pomocą matplotlib.
- **test.py**: Moduł z testami jednostkowymi, funkcjonalnymi, integracyjnymi i granacyjnymi, wykorzystujący unittest i memory_profiler.
- **data.json**: Plik przechowujący dane pokoi w formacie JSON.
- **test_data.json**: Tymczasowy plik używany w testach do zapisu i wczytywania danych.

Krótkie omówienie każdej klasy/modułu

- **Najemca**: Klasa przechowująca dane najemcy (imię, nazwisko, email) z metodą to_dict do konwersji na słownik JSON.
- **Pokoj**: Bazowa klasa dla pokoi, przechowuje numer, czynsz, najemcę i lokalizację (słownik z miastem, ulicą, numerem domu). Zawiera walidację dodatniego czynszu.
- **PremiumPokoj**: Klasa dziedzicząca po Pokoj, dodająca listę udogodnień (np. WiFi, TV).
- **App**: Główna klasa aplikacji, zarządzająca listą pokoi, interakcją z użytkownikiem i operacjami zapisu/odczytu danych. Implementuje menu, dodawanie, edytowanie, filtrowanie i wizualizację.
- **utils.py**: Funkcje load_data i save_data do obsługi plików JSON.
- **visualization.py**: Funkcja plot_rent_distribution generująca histogram czynszu.
- **test.py**: Testy weryfikujące poprawność klas, metod, obsługi błędów i wydajności.

4. Technologie i biblioteki

- **Python 3.10+**
- **json**: Do zapisu i wczytywania danych w formacie JSON.
- **functools**: Użycie reduce do operacji na czynszach i dekoratora @log_execution_time.
- **time**: Do pomiaru czasu wykonania metod.
- **os**: Do sprawdzania istnienia plików i ich usuwania w testach.
- **matplotlib**: Do wizualizacji rozkładu czynszu.
- **unittest**: Do testów jednostkowych, funkcjonalnych, integracyjnych i granicznych.
- **memory_profiler**: Do analizy zużycia pamięci w teście test_memory_save.

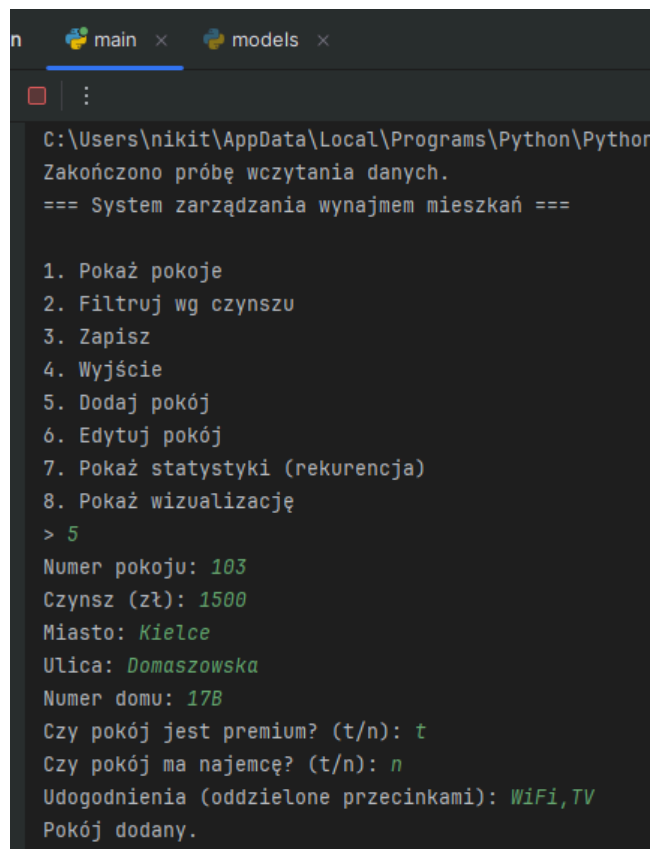
5. Sposób działania programu

Przykładowe dane wejściowe/wyjściowe

Dodawanie pokoju

=== 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
> 5
Numer pokoju: 103
Czynsz (zł): 1500
Miasto: Kielce
Ulica: Domaszowska
Numer domu: 17B
Czy pokój jest premium? (t/n): t
Czy pokój ma najemcę? (t/n): n
Udogodnienia (oddzielone przecinkami): WiFi,TV
Pokój dodany.
```

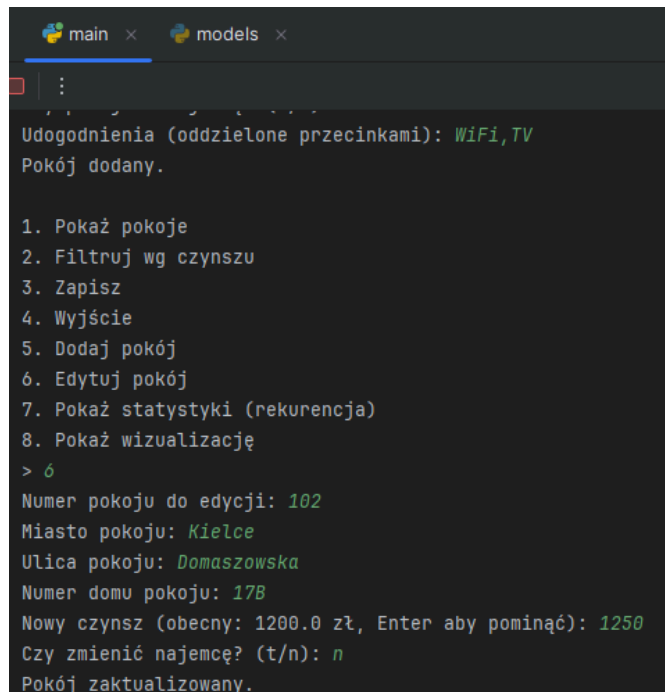


```
n  main x  models x
C:\Users\nikit\AppData\Local\Programs\Python\Python
Zakończono próbę wczytania danych.
=== System zarządzania wynajmem mieszkań ===

1. Pokaż pokoje
2. Filtruj wg czynszu
3. Zapisz
4. Wyjście
5. Dodaj pokój
6. Edytuj pokój
7. Pokaż statystyki (rekurencja)
8. Pokaż wizualizację
> 5
Numer pokoju: 103
Czynsz (zł): 1500
Miasto: Kielce
Ulica: Domaszowska
Numer domu: 17B
Czy pokój jest premium? (t/n): t
Czy pokój ma najemcę? (t/n): n
Udogodnienia (oddzielone przecinkami): WiFi,TV
Pokój dodany.
```

Edytowanie pokoju

1. Pokaż pokoje
 2. Filtruj wg czynszu
 3. Zapisz
 4. Wyjście
 5. Dodaj pokój
 6. Edytuj pokój
 7. Pokaż statystyki (rekurencja)
 8. Pokaż wizualizację
- > 6
- Numer pokoju do edycji: 102
Miasto pokoju: Kielce
Ulica pokoju: Domaszowska
Numer domu pokoju: 17B
Nowy czynsz (obecny: 1200.0 zł, Enter aby pominąć): 1350
Czy zmienić najemcę? (t/n): n
Pokój zaktualizowany.



```
main x models x
Udogodnienia (oddzielone przecinkami): WiFi,TV
Pokój dodany.

1. Pokaż pokoje
2. Filtruj wg czynszu
3. Zapisz
4. Wyjście
5. Dodaj pokój
6. Edytuj pokój
7. Pokaż statystyki (rekurencja)
8. Pokaż wizualizację
> 6
Numer pokoju do edycji: 102
Miasto pokoju: Kielce
Ulica pokoju: Domaszowska
Numer domu pokoju: 17B
Nowy czynsz (obecny: 1200.0 zł, Enter aby pominąć): 1250
Czy zmienić najemcę? (t/n): n
Pokój zaktualizowany.
```

Błąd przy edycji nieistniejącego pokoju

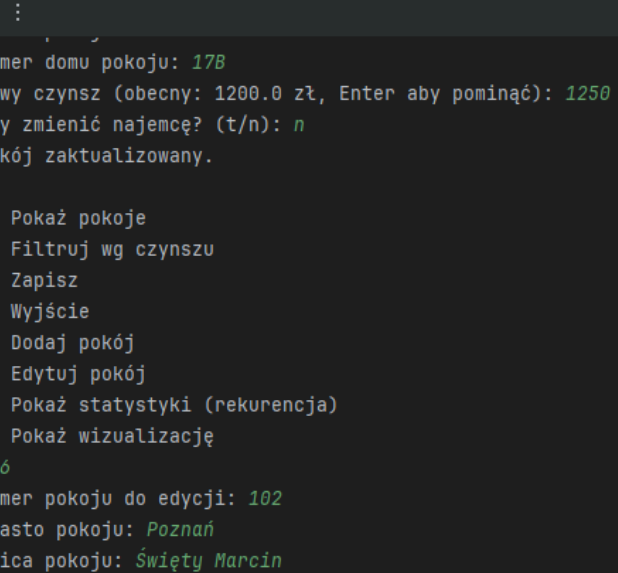
Numer pokoju do edycji: 102

Miasto pokoju: Poznań

Ulica pokoju: Święty Marcin

Numer domu pokoju: 15

Błąd: Pokój o tym numerze i lokalizacji nie istnieje!



```
Run main x models x
Numer domu pokoju: 17B
Nowy czynsz (obecny: 1200.0 zł, Enter aby pominąć): 1250
Czy zmienić najemcę? (t/n): n
Pokój zaktualizowany.

1. Pokaż pokoje
2. Filtruj wg czynszu
3. Zapisz
4. Wyjście
5. Dodaj pokój
6. Edytuj pokój
7. Pokaż statystyki (rekurencja)
8. Pokaż wizualizację
> 6
Numer pokoju do edycji: 102
Miasto pokoju: Poznań
Ulica pokoju: Święty Marcin
Numer domu pokoju: 5
Błąd: Pokój o tym numerze i lokalizacji nie istnieje!
```

Błąd przy dodaniu zduplikowanego pokoju

Numer pokoju: 102

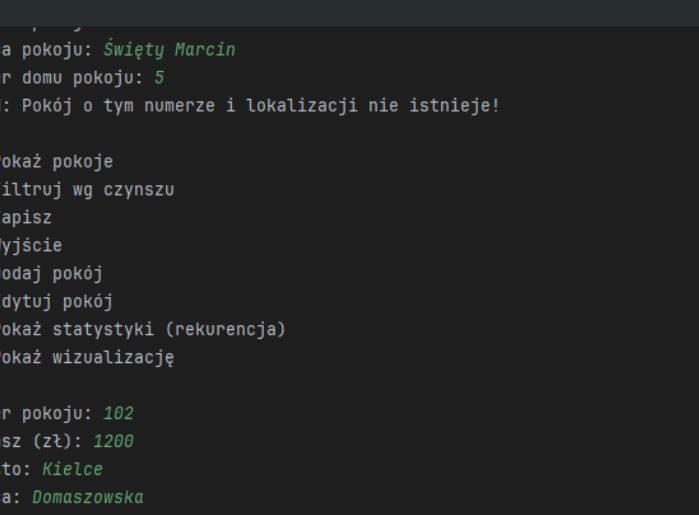
Czynsz (zł): 1200

Miasto: Kielce

Ulica: Domaszowska

Numer domu: 17B

Błąd danych wejściowych: Pokój o tym numerze i lokalizacji już istnieje!



```
Run main x models x
Ulica pokoju: Święty Marcin
Numer domu pokoju: 5
Błąd: Pokój o tym numerze i lokalizacji nie istnieje!

1. Pokaż pokoje
2. Filtruj wg czynszu
3. Zapisz
4. Wyjście
5. Dodaj pokój
6. Edytuj pokój
7. Pokaż statystyki (rekurencja)
8. Pokaż wizualizację
> 5
Numer pokoju: 102
Czynsz (zł): 1200
Miasto: Kielce
Ulica: Domaszowska
Numer domu: 17B
Błąd danych wejściowych: Pokój o tym numerze i lokalizacji już istnieje!
```

6. Przykłady kodu (z wyjaśnieniem)

models.py:

Fragment funkcji funkcyjnej

Funkcja `filter_czynsz` w klasie `App` wykorzystuje programowanie funkcyjne do filtrowania pokoi według maksymalnego czynszu:

```
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))
        if not znalezione:
            print("Brak pokoi w podanym limicie.")
        for p in znalezione:
            print(f"Pokój {p.numer}, Czynsz: {p.czynsz} zł")
    except ValueError:
        print("Nieprawidłowa kwota.")
```

Funkcja używa `filter` z wyrażeniem `lambda`, aby wybrać pokoje, których czynsz jest mniejszy lub równy podanemu limitowi. Obsługuje wyjątek `ValueError` dla nieprawidłowych danych wejściowych.

Fragment klasy

Klasa `PremiumPokoje` dziedziczy po `Pokoje` i dodaje obsługę udogodnień:

```
class Pokoje:
    """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

    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
        }

class PremiumPokoje(Pokoje):
    """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 []

    def to_dict(self):
```

```

    """Konwersja obiektu pokoju premium na słownik."""
    data = super().to_dict()
    data["udogodnienia"] = self.udogodnienia
    return data

```

Klasa dziedziczy atrybuty i metody klasy Pokoj, dodając listę udogodnienia. Metoda to_dict rozszerza metodę bazową, dodając pole udogodnienia do słownika JSON.

Obsługa wyjątków

Metod edit_pokoj zawiera obsługę błędów podczas edycji pokoju:

```

def edit_pokoj(self):
    """Edytowanie istniejącego pokoju."""
    try:
        numer = int(input("Numer pokoju do edycji: "))
        miasto = input("Miasto pokoju: ")
        ulica = input("Ulica pokoju: ")
        numer_domu = input("Numer domu pokoju: ")
        lokalizacja = {"miasto": miasto, "ulica": ulica, "numer_domu":
numer_domu}

        # Szukamy pokoju z podanym numerem i lokalizacją
        found = False
        for pokoj in self.pokoje:
            if pokoj.number == numer and pokoj.lokalizacja == lokalizacja:
                found = True
                czynsz = input(f"Nowy czynsz (obecny: {pokoj.czynsz} zł, Enter
aby pominąć): ")
                if czynsz:
                    pokoj.czynsz = float(czynsz)
                if input("Czy zmienić najemcę? (t/n): ").lower() == "t":
                    imie = input("Imię najemcy: ")
                    nazwisko = input("Nazwisko najemcy: ")
                    email = input("Email najemcy: ")
                    pokoj.najemca = Najemca(imie, nazwisko, email)
                if isinstance(pokoj, PremiumPokoj):
                    udogodnienia = input("Nowe udogodnienia (oddzielone
przecinkami, Enter aby pominąć): ")
                    if udogodnienia:
                        pokoj.udogodnienia = udogodnienia.split(",")
                        print("Pokój zaktualizowany.")
                        break
            if not found:
                raise AssertionError("Pokój o tym numerze i lokalizacji nie
istnieje!")
        except (ValueError, AssertionError) as e:
            print(f"Błąd: {e}")

```

Metoda sprawdza poprawność danych wejściowych (np. numer jako liczba całkowita) i unikalność pokoju (numer i lokalizacja). Wyjątki ValueError (dla nieprawidłowego typu danych) i AssertionError (dla nieistniejącego pokoju) są obsługiwane w bloku try-except.

7. Testowanie

Opis sposobu testowania

Testy zostały zaimplementowane w module test.py z użyciem biblioteki unittest. Obejmują:

- **Testy jednostkowe:** Sprawdzanie poprawności metod to_dict dla klas Najemca i Pokoj oraz atrybutów PremiumPokoje.
- **Testy funkcjonalne:** Test test_functional_filter_czynsz weryfikuje filtrowanie pokoi według czynszu.
- **Testy integracyjne:** Test test_integration_save_load sprawdza zapis i wczytywanie danych z pliku JSON.
- **Testy graniczne:** Testy test_invalid_czynsz, test_invalid_numer, test_invalid_lokalizacja i test_edit_pokoj_invalid_number_and_lokalizacja weryfikują obsługę błędnych danych (ujemny czynsz, zduplikowany numer/lokalizacja, nieistniejący pokój).
- **Test wydajności:** Test test_performance_save mierzy czas zapisu danych.
- **Test pamięci:** Test test_memory_save z użyciem memory_profiler analizuje zużycie pamięci.

Obsługa przypadków granicznych

- **Ujemny czynsz:** Test test_invalid_czynsz sprawdza, czy tworzenie pokoju z ujemnym czynszem powoduje AssertionError.
- **Zduplikowany numer i lokalizacja:** Test test_invalid_lokalizacja weryfikuje, że dodanie pokoju z istniejącym numerem i lokalizacją wywołuje błąd.
- **Nieistniejący pokój:** Test test_edit_pokoj_invalid_number_and_lokalizacja sprawdza, że próba edycji nieistniejącego pokoju (zły numer lub lokalizacja) powoduje błąd.
- **Zduplikowany numer z inną lokalizacją:** Test test_invalid_numer potwierdza, że można dodać pokój z tym samym numerem, ale inną lokalizacją.

8. Wnioski

Projekt pozwolił na zastosowanie pełnego zakresu materiału z języków skryptowych. Szczególnie wartościowe było połączenie Pythona z praktycznym zastosowaniem w zarządzaniu wynajmem mieszkań oraz tworzenie kodu wspierającego organizację danych o nieruchomościach. Aplikacja może stanowić podstawę większego systemu do zarządzania nieruchomościami, wspierającego np. agencje wynajmu lub platformy internetowe.

GitHub: <https://github.com/Nuggetsik/WynajemMieszkan>