Tematy projektów do wyboru

Nr	Tytuł projektu	Opis
1	~~System rezerwacji hoteli~~	~~Klasy Hotel, Room, Reservation; moduły, pliki JSON, dekoratory, filter, obsługa błędów, parsowanie stringów.~~
2	Zarządzanie projektami zespołowymi (Kanban)	Zadania, użytkownicy, statusy – klasy, wyjątki, przestrzenie nazw, moduły, analiza przez map, filter.
3	~~Planer posiłków z generatorem list zakupów~~	~~Przepisy jako obiekty, dane z plików, filter po składnikach, asercje, przestrzenie nazw, funkcjonalność API lokalnego.~~
4	~~System ocen i statystyk w szkole~~	~~Klasy Student, Grade, Subject, pliki CSV, średnie przez reduce, wyjątki i walidacja danych, przestrzenie nazw.~~
5	~~Gra tekstowa RPG~~	~~Gracze, ekwipunek, walka, klasy, funkcje, przestrzenie nazw, pliki save, przetwarzanie stringów, błędy, lambda.~~
6	~~System biblioteczny z rezerwacją książek~~	~~Wyszukiwanie po tytule/autorce, klasy, moduły, obsługa wyjątków, iteracje po rekordach, operacje plikowe.~~
7	~~Kreator i odtwarzacz quizów edukacyjnych~~	~~Tworzenie quizów, klasy Question, Quiz, pliki .json, obsługa błędów, analiza wyników przez map, filter.~~
8	~~Budżet osobisty z analizą wydatków~~	~~Dane w słownikach, raporty miesięczne, klasy Transaction, Budget, operacje na plikach, lambda, wyjątki.~~
9	Kalendarz z przypomnieniami i wydarzeniami	Praca z datami, stringami, klasy Event, operacje plikowe, przestrzenie nazw, funkcje użytkownika, filter.
10	System zgłoszeń technicznych (Helpdesk)	Obsługa statusów, użytkownicy, klasy, błędy, wpisy do plików, struktury danych, assert, dekoratory logujące.
11	~~Symulator banku z kontami i przelewami~~	~~Klasy Account, User, wyjątki (np. debet), operacje na plikach, filter transakcji, walidacja i asercje.~~
12	~~Zarządzanie drużyną sportową i wynikami~~	~~Klasy Player, Team, Match, analiza wyników, zapis do CSV, przestrzenie nazw, stringi i wyjątki.~~
13	~~Szyfrator wiadomości tekstowych~~	~~Operacje na stringach, klasy Cipher, zapis/odczyt, obsługa błędów, pakiety szyfrujące, lambda.~~
14	~~System głosowania (np. wybory samorządu studenckiego)~~	~~Klasy, pliki wyników, wyjątki, przestrzenie nazw, analiza przez funkcje funkcyjne (groupby, filter).~~
15	~~Zarządzanie firmą kurierską (zlecenia, trasy)~~	~~Klasy Package, Driver, Route, zapis trasy do pliku, analiza czasu przez map, obsługa błędów.~~
16	~~Planer nauki do egzaminów~~	~~Zadania, daty, analiza przez lambda, zapis planu do pliku, klasy Task, przestrzenie nazw, wyjątki.~~
17	System wypożyczalni rowerów/skuterów	Zasoby, użytkownicy, raporty, zapis stanu, przetwarzanie danych, filter, assert, klasy i pliki.
~~18~~	~~Kreator CV i listów motywacyjnych~~	~~Operacje na stringach, klasy Profile, CVBuilder, pliki .txt/.pdf, błędy użytkownika, przestrzeń nazw.~~
19	Rozkład jazdy i planer podróży komunikacją	Trasy, godziny, wyszukiwanie przez filter, struktury danych, wyjątki, klasy Stop, Line.
20	~~Gra w statki (tekstowa)~~	~~Plansze jako listy, funkcje, klasy Board, Ship, przestrzenie nazw, zapis do pliku, dekoratory logujące.~~
21	~~System śledzenia celów i postępów~~	~~Cele jako klasy, lista celów jako kontener, przestrzenie nazw, lambda do filtrowania celów, pliki, wyjątki.~~
22	~~Program „dziennik zdrowia” z analizą BMI~~	~~Dane z pliku, klasy User, Entry, analiza lambda, wyjątki i walidacje, przetwarzanie stringów.~~
23	Symulator lotniska (loty, terminale)	Klasy Flight, Terminal, wyjątki, sortowanie i filter lotów, pliki jako logi systemowe.
24	~~Gra „wisielec” z interaktywnym słownikiem~~	~~Klasy Game, Word, operacje na stringach, losowanie, pliki z wynikami, wyjątki, przestrzenie nazw.~~
25	~~System do analizy danych meteorologicznych~~	~~Odczyt CSV, klasy Record, Station, analiza map, reduce, pliki, wyjątki, przestrzeń nazw i pakiety.~~

26	~~System zarządzania kartoteką medyczną~~	~~Klasy Patient, Visit; zapis danych do plików; przetwarzanie tekstu; filter() po objawach; wyjątki i moduły.~~
27	~~Aplikacja do monitorowania snu~~	~~Klasy SleepSession, dane w pliku, analiza przez lambda; wizualizacja konsolowa; obsługa błędów, przestrzenie nazw.~~
28	~~Katalog filmów z ocenami i recenzjami~~	~~Obiekty Movie, Review; pliki JSON; analiza ocen przez map/reduce; operacje na stringach i wyjątki użytkownika.~~
29	~~Symulator stacji benzynowej~~	~~Obsługa transakcji, klasy FuelPump, Transaction; błędy (brak paliwa), zapis historii, średnia cena map().~~
30	System zarządzania wynajmem mieszkań	Dane najemców, pokoje; filter wg czynszu; pliki; klasy; przestrzenie nazw; obsługa błędów.
31	~~Trener matematyczny – quiz + analiza postępów~~	~~Operacje na stringach, klasy, błędy przy dzieleniu, filter po wynikach, pliki CSV z wynikami, pakiety z quizami.~~
32	Planer świąteczny z budżetem i listą prezentów	Listy, słowniki, operacje na danych, klasy Gift, zapis wydatków, filter() po osobie, wyjątki – brak budżetu.
33	Lokalna sieć wymiany książek	Katalog książek, użytkownicy, klasy, zapis stanu do pliku, filtrowanie wg autora, przestrzenie nazw i moduły.
34	~~System do rejestracji czasu pracy~~	~~Employee, WorkEntry, analiza czasu pracy, operacje na plikach, wyjątki przy błędnych datach, lambda.~~
35	~~Aplikacja do nauki znaków drogowych~~	~~Klasy znaków, quiz, zapis wyników, analiza przez filter, stringi (opisy), wyjątki (błędne dane wejściowe).~~
36	~~System opieki nad zwierzętami w schronisku~~	~~Zwierzęta jako obiekty, adopcje, pliki danych, filter, map, błędy, zarządzanie klasami i modułami.~~
37	~~Aplikacja do zarządzania zużyciem energii w domu~~	~~Klasy Device, dane w CSV, analiza przez reduce, asercje na limity, zapis wyników do plików.~~
38	System generowania ofert pracy	Oferty, kandydaci, klasy, sortowanie i filtrowanie, stringi w opisach, przestrzeń nazw, zapis do plików.
39	~~Aplikacja do nauki alfabetu Morse'a~~	~~Kodowanie/odkodowywanie, stringi, błędy przy nieznanych znakach, assert, klasy MorseEncoder.~~
40	~~Symulator turnieju piłkarskiego~~	~~Drużyny, mecze, klasy, analiza wyników, obsługa wyjątków, map, zapis wyników do pliku.~~
41	Asystent zakupowy z porównywarką cen	Product, Store, sortowanie po cenie, filter po kategorii, zapis koszyka, błędy danych wejściowych.
42	System rejestracji uczestników konferencji	Klasy Attendee, Event, obsługa miejsc, zapis danych, operacje na stringach, wyjątki (limit przekroczony).
43	~~Program do planowania treningów siłowych~~	~~Exercise, WorkoutPlan, analiza przez lambda, pliki, błędy (np. brak danych), klasy i przestrzenie nazw.~~
44	Kalkulator ekologicznego śladu węglowego	Dane wejściowe użytkownika, analiza przez map, klasy, zapis do plików, wyjątki, operacje tekstowe.
45	~~System do monitorowania poziomu nawodnienia~~	~~User, HydrationEntry, analiza średniego spożycia wody, zapis do plików, błędy, filtracja, klasy.~~
46	Aplikacja do tworzenia tygodniowych jadłospisów	Klasy Meal, plan jako kontener, operacje plikowe, filter po typie posiłku, wyjątki, przestrzenie nazw.
47	System monitorowania ogrodu (czujniki, pogoda)	Klasy Sensor, Plant, Reading, analiza danych, pliki CSV, wyjątki i przestrzeń nazw, map/filter.
48	Asystent podróży z notatnikiem trasy	Klasy Trip, Note, stringi i daty, błędy w danych, zapis do JSON, lambda do wyszukiwania.
49	System planowania zadań dla małego zespołu	Task, TeamMember, pliki, wyjątki (brak przypisania), funkcje funkcyjne do analizy zadań.
50	Aplikacja wspierająca naukę języka migowego	Stringi, symbole, quiz, klasy Gesture, Lesson, wyjątki przy błędach w pliku, analiza postępów.
51	Aplikacja do nauki Braille'a z generowaniem G-code do druku 3D	Stringi, symbole, quiz, klasy Gesture, Lesson, wyjątki przy błędach w pliku, analiza postępów.
52	~~Katalog gier z recenzjami i ocenami~~	~~Obiekty Game, Review; pliki JSON; analiza ocen przez map/reduce; operacje na stringach i wyjątki użytkownika.~~
53	System zarządzania i kontroli ruchu	

	dronów osobowych i towarowych w inteligentnym mieście przyszłości	
54	Preper - System zarządzania zapasami z generowaniem kodów ich wydrukiem i odczytem	Informacje o zapasach o ich umiejscowieniu, o dacie przydatności do spożycia, kaloryczności. Itp. Zapis danych do pliku. Odczyt danych z pliku, generowanie etykiet, Generowanie powiadomień o zbliżających się terminach przydatności i zaistniałych brakach.
55	Preper - System zarządzania lekami i środkami opatrunkowymi. Informacjami o lekach i ziołach leczniczych	
56	Preper - System Informacji o ziołach leczniczych oraz sposobie przyrządzania z nich leków.	
57	Preper - System Informacji o dziko rosnących roślinach jadalnych oraz sposobie przyrządzania z nich pożywienia	
58	~~Gra komputerowa typu "Flappy Bird".~~	
59	~~Chatbot wykorzystujący technologię ChatGPT lub Deepseek~~	
60	~~Gra komputerowa typu „wąż”~~	
61	~~Gar komputerowa typu Statki wraz z serwerem TCP/IP~~	

UWAGA Wykreślone tematy są już wybrane przez inne zespoły !!!

System rezerwacji pokoi hotelowych z analizą statystyczną i API do zarządzania

Co ma zawierać projekt:

1. **Wprowadzenie do języków skryptowych** – projekt pisany w Pythonie jako języku skryptowym.
2. **Podstawy** – zmienne, instrukcje warunkowe, pętle (np. iteracja po rezerwacjach), wejście/wyjście (terminalowe i plikowe).
3. **Kontenery danych** – listy rezerwacji, słowniki użytkowników, zbiór dostępnych pokoi.
4. **Przestrzenie i zasięgi nazw** – funkcje wewnętrzne, prywatne metody w klasach, zmienne lokalne i globalne.
5. **Moduły i pakiety** – podział systemu na moduły: `gui.py`, `logic.py`, `data.py`, `api.py`.
6. **Obsługa błędów (wyjątki, asercje)** – np. `assert price > 0`, `try...except` przy plikach, błędach rejestracji.
7. **Łańcuchy znaków** – filtrowanie rezerwacji po nazwie klienta, budowanie komunikatów, parsowanie dat.
8. **Obsługa plików** – zapis danych rezerwacji do plików `.json`, odczyt, aktualizacja.
9. **Programowanie obiektowe** – klasy `Hotel`, `Room`, `Reservation`, dziedziczenie np. `VIPRoom(Room)`.
10. **Programowanie funkcyjne** – użycie `filter()`, `map()`, `lambda`, `reduce()` do analizy statystyk (np. średnia cena za noc).

System zarządzania projektami zespołowymi (Kanban)

- Wszystkie dane trzymane w plikach `.json`.
- Zadania jako obiekty z metodami (`Task`, `User`, `Project`).
- Kategorie, statusy, filtrowanie po tagach – `filter`, `lambda`.
- Obsługa wyjątków: nieprawidłowy status, puste dane, duplikaty.
- Moduł `data_handler.py` do operacji plikowych.
- Reprezentacja zadań jako stringi z formatowaniem.
- Asercje: np. że każda task ma przypisanego użytkownika.

- Użycie przestrzeni nazw – oddzielenie GUI od logiki.
- Zautomatyzowane testy przy pomocy `assert`.

Inteligentny planer posiłków i zakupów

- Użytkownik wybiera posiłki, tworzy plan, system wylicza składniki i generuje listę zakupów.
- Funkcjonalności:

    - Operacje na stringach – parsowanie receptur.
    - Kontenery – lista przepisów, słownik składników.
    - Programowanie funkcyjne – `map()`, `filter()` do analizy diety.
    - Moduły: `planner.py`, `recipes.py`, `storage.py`.
    - Obsługa wyjątków – błędy odczytu plików, brak składników.
    - Programowanie obiektowe – klasy `Meal`, `UserPlan`, `ShoppingList`.
    - Przestrzeń nazw i zakres zmiennych: lokalne/klasowe/metodowe.
    - Asercje przy walidacji wartości odżywczych.
    - Tworzenie i obsługa plików `.csv`, `.json`.

Harmonogram 4-tygodniowy dla każdego projektu

Tydzień	Zadania	Rezultaty
1	Analiza tematu, podział ról, zaplanowanie architektury, rozpoczęcie pracy nad klasami i funkcjami pomocniczymi.	Dokument z analizą wymagań, szkic diagramu klas, rozpoczęty plik `main.py`.
2	Implementacja głównej logiki, obsługa błędów, kontenery danych, podstawowa wersja operacji I/O i zapis do pliku.	Działająca logika programu (konsola lub podstawowy interfejs), zapis danych do pliku `data.json`.
3	Dodanie programowania funkcyjnego, testów, przetwarzania stringów, separacja modułów (`utils.py`, `models.py`), asercje.	Refaktoryzacja kodu, pełna funkcjonalność, testowanie.
4	Dokumentacja, testy końcowe, przygotowanie prezentacji projektu i kodu.	Raport zaliczeniowy, uporządkowany kod, instrukcja uruchomienia, gotowość do oceny.

Wzór dokumentacji i raportu zaliczeniowego

1. Strona tytułowa

- Nazwa uczelni
- Nazwa przedmiotu
- Tytuł naukowy oraz imię i nazwisko prowadzącego zajęcia
- Tytuł projektu
- Imiona i nazwiska członków zespołu
- Grupa studencka
- Data oddania

2. Opis projektu

- Cel projektu
- Funkcje aplikacji
- Zakres funkcjonalny (co zostało zaimplementowane)

3. Struktura projektu

- Opis plików i folderów
- Krótkie omówienie każdej klasy/modułu

4. Technologie i biblioteki

- Python 3.x
- Dodatkowe: json, datetime, os, ewentualnie tkinter, functools, itertools itd. Itp.

5. Sposób działania programu

- Instrukcja uruchomienia
- Przykładowe dane wejściowe/wyjściowe
- Zrzuty ekranu (jeśli GUI)

6. Przykłady kodu (z wyjaśnieniem)

- Fragment funkcji funkcyjnej
- Fragment klasy
- Obsługa wyjątków

7. Testowanie

- Opis sposobu testowania
- Obsługa przypadków granicznych

8. Wnioski

- Co się udało
- Co można było zrobić lepiej
- Jakie kompetencje zostały rozwinięte

Przykładowy szablon

Projekt zaliczeniowy – Języki skryptowe (Python)

Tytuł projektu: Aplikacja do nauki Braille'a z generowaniem G-code do druku 3D
Autorzy: Jan Kowalski, Anna Nowak
Grupa: Informatyka, rok II
Data oddania: _____

1. Cel projektu

Celem projektu było stworzenie aplikacji umożliwiającej konwersję tekstu wprowadzonego przez użytkownika na alfabet Braille'a oraz automatyczne wygenerowanie pliku `.gcode` do wydruku wypukłego tekstu na drukarce 3D.

2. Zakres funkcjonalny

- Wprowadzanie tekstu przez użytkownika z poziomu terminala
- Konwersja każdej litery na jej odpowiednik w układzie Braille'a (system 6-punktowy)
- Generowanie ścieżek `G-code` dla każdego punktu wypukłego
- Obsługa błędów (np. nieobsługiwane znaki)
- Zapis pliku `.gcode` gotowego do eksportu do drukarki 3D

3. Struktura projektu

- `main.py`: główny moduł uruchamiający aplikację
- `braille_converter.py`: funkcje przetwarzające tekst na tablice kropek
- `gcode_generator.py`: funkcje do generowania kodu G-code

- `output/`: folder wyjściowy, zawierający plik `example_output.gcode`

- `README.md`: opis działania projektu

4. Technologie i biblioteki

- Python 3.x
- Obsługa plików tekstowych
- Podstawy G-code (`G0`, `G1`)
- Opcjonalnie: `FPDF` do raportów PDF

5. Programowanie obiektowe i funkcyjne

- Użycie klas i mapowań (`dict`) do konwersji znaków
- Filtrowanie danych za pomocą `filter()` i `lambda`
- Moduły `braille_converter` i `gcode_generator` wydzielone dla lepszej organizacji
- Użycie przestrzeni nazw dla separacji logiki

6. Obsługa błędów i plików

- Zabezpieczenie przed wczytaniem błędnych znaków
- Obsługa wyjątków przy odczycie/zapisie plików
- Walidacja danych wejściowych
- Automatyczne tworzenie katalogu `output` i pliku `.gcode`
- Testy oprogramowania

7. Wnioski

Projekt pozwolił na zastosowanie całego zakresu materiału z języków skryptowych. Szczególnie wartościowe było połączenie Pythona z praktycznym zastosowaniem w technologii druku 3D oraz tworzenie kodu generującego fizyczny efekt. Aplikacja może być podstawą większego systemu wspomagającego naukę języka Braille'a dla osób niedowidzących.

Szablon struktury plików + pliki startowe

/projekt/

|

├── main.py # Punkt wejścia – menu główne, sterowanie programem

├── utils.py # Funkcje pomocnicze

├── models.py # Klasy obiektowe

├── data.json # Dane użytkownika / zapis stanu

├── README.md # Instrukcja

└── raport.pdf # Raport końcowy

main.py

```
from utils import load_data, save_data
from models import App

def main():
    app = App()
    app.run()

if __name__ == "__main__":
    main()
```

utils.py

```
import json

def load_data(filename="data.json"):
    try:
        with open(filename, "r") as f:
            return json.load(f)
    except FileNotFoundError:
        return {}

def save_data(data, filename="data.json"):
    with open(filename, "w") as f:
        json.dump(data, f, indent=4)
```

models.py

```
class App:
    def __init__(self):
        self.data = {}

    def run(self):
```

```
        print("Witaj w aplikacji!")
        # przykład użycia funkcji funkcyjnej
        self.menu()

    def menu(self):
        while True:
            choice = input("1. Start\n2. Wyjście\n> ")
            if choice == "1":
                print("Funkcja startowa...")
            elif choice == "2":
                break
            else:
                print("Nieprawidłowy wybór")

        assert isinstance(self.data, dict), "Dane muszą być słownikiem"
```

`data.json` (przykład)

```
{
  "users": [],
  "projects": [],
  "settings": {
    "theme": "light"
  }
}
```