Podstawowe operacje na plikach

1. Otwieranie i zamykanie pliku:

Używamy funkcji `open()`, która przyjmuje dwa parametry: nazwę pliku oraz tryb otwarcia.

```
file = open("nazwa_pliku.txt", "r")
file.close()
```

Aby automatycznie zamknąć plik, można użyć konstrukcji `with`, co jest zalecane, ponieważ zapewnia zamknięcie pliku po zakończeniu pracy.

```
with open("nazwa_pliku.txt", "r") as file:
    # Operacje na pliku
```

2. Tryby otwarcia pliku

- `"r"` - odczyt
- `"w"` - zapis (usuwa zawartość istniejącego pliku)
- `"a"` - dopisanie (dodaje dane na końcu istniejącego pliku)
- `"r+"` - odczyt i zapis
- `"w+"` - zapis i odczyt (tworzy nowy plik, jeśli nie istnieje)
- `"a+"` - dopisanie i odczyt

3. Odczyt danych z pliku

- `read()` - wczytuje całą zawartość pliku jako jeden ciąg znaków.
- `readline()` - wczytuje jedną linię na raz.
- `readlines()` - wczytuje wszystkie linie jako listę, gdzie każda linia jest jednym elementem listy.

```
with open("nazwa_pliku.txt", "r") as file:
    data = file.read()
    lines = file.readlines()
```

4. Zapis do pliku

- `write()` - zapisuje dane do pliku.
- `writelines()` - zapisuje listę linii do pliku.

```
with open("nazwa_pliku.txt", "w") as file:
    file.write("Nowa linia tekstu\n")
    file.writelines(["Linia 1\n", "Linia 2\n"])
```

5. Praca z plikami binarnymi

Przy pracy z plikami binarnymi, np. obrazami, używamy trybu `"rb"` (odczyt binarny) lub `"wb"` (zapis binarny).

```
with open("obraz.jpg", "rb") as file:
    content = file.read()
    binary_data = file.read(100)

with open("kopiowany_obraz.jpg", "wb") as file:
    file.write(content)
```

6. Przemieszczanie się po pliku

- `seek(offset)` - ustawia wskaźnik na konkretną pozycję.
- `tell()` - zwraca aktualną pozycję wskaźnika.

```
with open("nazwa_pliku.txt", "r") as file:
    file.seek(10)
    print(file.tell())
    print(file.readline())
```

7. Usuwanie pliku

Możemy usunąć plik za pomocą modułu `os`.

```
import os
os.remove("nazwa_pliku.txt")
```

Bardziej zaawansowane operacje na plikach

1. Praca z dużymi plikami – przetwarzanie strumieniowe

Przetwarzanie dużych plików w małych porcjach jest ważne, aby nie zużyć całej pamięci RAM. Można to zrobić, czytając plik linia po linii lub określoną liczbę bajtów na raz.

```
with open("duzy_plik.txt", "r") as file:
    for line in file:
        process(line)
```

2. Zapis i odczyt danych w formacie JSON

Dane `json` można łatwo zapisywać i odczytywać, co jest przydatne przy przechowywaniu złożonych struktur danych.

- `.dump()` - serializacja do formatu JSON.
- `.load()` - deserializacja do obiektu języka Python.

```
import json

data = {"name": "Jan", "age": 30, "city": "Warszawa"}
with open("dane.json", "w") as file:
    json.dump(data, file)

with open("dane.json", "r") as file:
    data = json.load(file)
    print(data)
```

3. Obsługa plików CSV

Python ma moduł `csv`, który ułatwia odczyt i zapis danych tabelarycznych.

- `csv.writer(file)` - tworzy obiekt do zapisywania danych w formacie CSV do podanego pliku.
- `writer.writerow()` - zapisuje pojedynczy wiersz danych (lista) w pliku CSV.
- `csv.reader(file)` - tworzy obiekt do odczytywania danych z pliku CSV.

Alternatywą dla tego podejścia jest biblioteka `pandas` umożliwiająca zaawansowane operacje na danych.

```
import csv

with open("dane.csv", "w", newline="") as file:
    writer = csv.writer(file)
    writer.writerow(["name", "age", "city"])
    writer.writerow(["Jan", 30, "Warszawa"])

with open("dane.csv", "r") as file:
    reader = csv.reader(file)
    for row in reader:
        print(row)
```

4. Obsługa plików ZIP

Moduł `zipfile` umożliwia tworzenie archiwów ZIP oraz wypakowywanie ich zawartości.

```python
import zipfile

with zipfile.ZipFile("archiwum.zip", "w") as archive:
    archive.write("dane.json")
    archive.write("dane.csv")

with zipfile.ZipFile("archiwum.zip", "r") as archive:
    archive.extractall("rozpakowane_pliki")
```

5. Praca z modułem `shutil`

`shutil` umożliwia zaawansowane operacje na plikach i katalogach, takie jak kopiowanie, przenoszenie, usuwanie całych katalogów.

Nawet funkcje kopiowania plików wyższego poziomu (shutil.copy(), shutil.copy2()) nie są w stanie skopiować wszystkich metadanych plików.

- `.copy()` - kopiowanie.
- `.move()` - przenoszenie.
- `rmtree()` - usuwanie całego katalogu wraz z jego zawartością.

```python
import shutil

shutil.copy("dane.csv", "kopia_dane.csv")

shutil.move("dane.csv", "przeniesione_dane.csv")

shutil.rmtree("rozpakowane_pliki")
```

6. Wykorzystanie `pathlib` do obsługi ścieżek plików

Moduł `pathlib` zapewnia wygodniejsze i bardziej przejrzyste operacje na ścieżkach plików.

- `.name` - zwraca nazwę pliku wraz z rozszerzeniem.
- `.parent` - zwraca katalog w którym znajduje się plik.
- `.suffix` - zwraca rozszerzenie pliku.

```python
from pathlib import Path

plik = Path("dane.csv")

if plik.exists():
    print(f"{plik} istnieje.")

print("Nazwa pliku:", plik.name)
print("Katalog:", plik.parent)
print("Rozszerzenie:", plik.suffix)
```

7. Przetwarzanie linii za pomocą `yield`

Aby przetwarzać bardzo duże pliki można użyć generatorów, aby uniknąć ładowania wszystkiego naraz do pamięci.

`yield` działa jako sposób na tworzenie generatorów w Pythonie. Generatory to funkcje, które zamiast zwracać całą wartość naraz (jak `return`), zatrzymują swój stan i zwracają wartości po jednej, umożliwiając iterację.

```
def read_large_file(file_name):
    with open(file_name, "r") as file:
        for line in file:
            yield line


for line in read_large_file("duzy_plik.txt"):
    print(line)
```

8. Tymczasowe pliki z tempfile

Moduł `tempfile` umożliwia tworzenie plików i katalogów tymczasowych, które mogą być używane do przechowywania danych tymczasowych.

Tymczasowe pliki tworzone przez moduł `tempfile` są zapisywane w systemowym katalogu plików tymczasowych. Lokalizacja tego katalogu zależy od systemu operacyjnego i konfiguracji środowiska.

```
import tempfile

with tempfile.TemporaryFile() as temp_file:
    temp_file.write(b"To jest plik tymczasowy")
    temp_file.seek(0)
    print(temp_file.read())
```

9. Peklowanie

Proces serializacji obiektów, czyli zapisywania ich w postaci, która pozwala na ich późniejsze odtworzenie. Python umożliwia serializację obiektów do formatu binarnego za pomocą modułu `pickle`, co jest przydatne, gdy chcemy przechować złożone struktury danych, takie jak listy, słowniki czy obiekty klas, i przywrócić je później w identycznej postaci.

Moduł `pickle` pozwala na:

- Serializację – konwersję obiektu do formatu binarnego (lub tekstowego w Pythonie 2), który można zapisać do pliku.
- Deserializację – odtworzenie obiektu z formatu binarnego do oryginalnej postaci.

```python
import pickle

dane = {"name": "Jan", "age": 30, "city": "Warszawa"}

with open("dane.pkl", "wb") as file:
    pickle.dump(dane, file)


with open("dane.pkl", "rb") as file:
    odczytane_dane = pickle.load(file)

print(odczytane_dane)
```

Zadania do realizacji

Zadanie 1: Analiza logów systemowych

1. Wczytaj plik `system_logs.txt` linia po linii.
2. Policz liczbę zdarzeń każdego typu (`INFO`, `WARNING`, `ERROR`) i wypisz wyniki na ekranie.
3. Zapisz wszystkie zdarzenia typu `ERROR` do nowego pliku `errors.log`.

Rozszerzenie Znajdź godzinę, w której wystąpiło najwięcej zdarzeń, i zapisz ją w pliku `summary.log`.

Zadanie 2: Archiwizacja plików

1. W pliku ZIP `archive.zip` znajdują się pliki tekstowe.
2. Rozpakuj plik ZIP do katalogu o nazwie `rozpakowane`.
3. Sprawdź rozmiar każdego pliku w katalogu `rozpakowane` i wypisz je w formacie: `Nazwa pliku - Rozmiar (bajtów)`.

Rozszerzenie Zarchiwizuj wszystkie pliki z katalogu `rozpakowane` do nowego pliku ZIP o nazwie `new_archive.zip`.

Zadanie 3: Praca z danymi JSON

1. Wczytaj dane z pliku `products.json`.
2. Wylicz łączną wartość produktów (cena × ilość) w każdej kategorii.
3. Zapisz wyniki w nowym pliku `category_summary.json` w formacie:

```json
{
    "Kategoria 1": 500.0,
    "Kategoria 2": 500.0
}
```

Rozszerzenie Znajdź najdroższy produkt i zapisz jego nazwę w pliku `most_expensive_product.txt`.

Zadanie 4: Analiza danych studentów w CSV

1. Wczytaj dane z pliku `students.csv`.
2. Oblicz średnią ocen wszystkich studentów i wypisz ją na ekranie.
3. Stwórz nowy plik CSV `students_summary.csv`, w którym dodasz kolumnę `Status`:
   - Jeśli średnia ocena studenta jest większa lub równa 3.0, ustaw `Status` na `Zaliczony`.
   - W przeciwnym razie ustaw `Status` na `Nie zaliczony`.

Rozszerzenie Posortuj dane w nowym pliku CSV alfabetycznie według imion studentów.

Zadanie 5: Serializacja obiektów za pomocą Pickle

```
class Pracownik:
    def __init__(self, imie, wiek, stanowisko):
        self.imie = imie
        self.wiek = wiek
        self.stanowisko = stanowisko

    def __repr__(self):
        return f"Imię: {self.imie}, Wiek: {self.wiek}, Stanowisko:
{self.stanowisko}"
```

1. W pliku `pracownicy.pkl` znajdują się obiekty klasy `Pracownik`.
2. Wczytaj plik i wyświetl szczegóły każdego pracownika w formacie:

```
Imię: Jan, Wiek: 30, Stanowisko: Inżynier
```

3. Dodaj nowego pracownika, a następnie zapisz zaktualizowaną listę pracowników z powrotem do pliku `pracownicy.pkl`.

Rozszerzenie Znajdź pracownika o najdłuższym stażu (najstarszego) i zapisz jego dane w pliku `senior_pracownik.pkl`.

Zadanie 6: Analiza sprzedaży z pliku CSV

1. Wczytaj dane z pliku `sales.csv`, gdzie każda linia zawiera: `Produkt, Ilość, Cena`.
2. Oblicz łączną sprzedaż dla każdego produktu (Ilość × Cena) i wypisz wyniki w formacie:

```
Produkt A: 300.0
Produkt B: 50.0
Produkt C: 1000.0
```

3. Zapisz wyniki w nowym pliku `sales_summary.csv`.

Rozszerzenie Znajdź produkt o najwyższej sprzedaży i zapisz jego nazwę w pliku `top_selling_product.txt`.

Uwaga

- **Pliki wymagane do realizacji zadań** znajdują się w archiwum **student_tasks.zip**
- Każde zadanie można rozwiązywać niezależnie.
- **Rozszerzenia** są opcjonalne i przeznaczone dla tych, którzy ukończyli podstawowe wersje zadań.

Zadanie dla chętnych

System zarządzania zadaniami z kolejką priorytetową i logowaniem

Opis problemu:
Program ma zarządzać kolejką zadań z różnymi priorytetami. Jego funkcjonalność obejmuje dodawanie zadań, przetwarzanie ich w kolejności priorytetów, usuwanie zadań na podstawie nazwy, a także zapisywanie i odczytywanie stanu kolejki za pomocą mechanizmu serializacji (`pickle`). Wszystkie akcje powinny być logowane w pliku `system_logs.txt`.

Wymagania:

1. **Klasa `Zadanie`**:
   - Powinna zawierać atrybuty:
     - `nazwa` - nazwa zadania, np. "Naprawa serwera".
     - `priorytet` - liczba całkowita oznaczająca priorytet (niższa liczba oznacza wyższy priorytet).
     - `czas_dodania` - czas utworzenia zadania, przypisywany automatycznie przy tworzeniu obiektu.
   - Powinna umożliwiać porównywanie obiektów na podstawie priorytetu, a w przypadku równości na podstawie czasu dodania (`__lt__`).
   - Reprezentacja tekstowa zadania powinna mieć format:

```
Zadanie: Naprawa serwera, Priorytet: 1, Dodano: 2024-11-11 12:00:00
```

2. **Klasa `SystemZadan`**:
   - Powinna zarządzać kolejką zadań za pomocą `heapq`.
   - Powinna oferować następujące funkcjonalności:
     - Dodanie zadania z priorytetem.
     - Przetwarzanie zadania o najwyższym priorytecie.

    - Usuwanie zadania na podstawie jego nazwy.
    - Wyświetlanie aktualnego stanu kolejki.
    - Zapis i odczyt stanu kolejki przy użyciu `pickle`.
  - Wszystkie akcje powinny być rejestrowane w logach.
3. **Logowanie**:
  - Plik `system_logs.txt` powinien rejestrować każdą akcję w formacie:

```
[2024-11-11 12:00:00] INFO Dodano zadanie: Naprawa serwera, Priorytet: 1
[2024-11-11 12:05:00] INFO Przetworzono zadanie: Naprawa serwera
[2024-11-11 12:10:00] INFO Zapisano stan kolejki
[2024-11-11 12:15:00] WARNING Próba usunięcia zadania, które nie istnieje: Zadanie A
```

Zakres do realizacji:

1. Utworzenie klasy `Zadanie` z odpowiednimi atrybutami i metodami, umożliwiającej porównywanie i reprezentowanie obiektów.
2. Implementacja klasy `SystemZadan`, obsługującej kolejkę priorytetową, z metodami do zarządzania zadaniami oraz zapisu i odczytu kolejki z pliku.
3. Dodanie logowania działań programu do pliku `system_logs.txt`.
4. Przetestowanie funkcji programu na przykładach, takich jak dodawanie, przetwarzanie i usuwanie zadań oraz zapis i odczyt stanu kolejki.

Dodatkowe wymagania (++opcjonalne):

1. Ograniczenie liczby zadań w kolejce do 10. Przy dodaniu nowego zadania w pełnej kolejce powinno być automatycznie usunięte zadanie o najniższym priorytecie. Działanie to powinno być zarejestrowane w logach.
2. Dodanie funkcji umożliwiającej analizę logów, np. wyświetlenie wszystkich ostrzeżeń (`WARNING`) lub informacji (`INFO`).
3. Stworzenie prostego interfejsu tekstowego pozwalającego na interakcję z użytkownikiem, z opcjami takimi jak: dodanie zadania, przetworzenie zadania, wyświetlenie kolejki czy zapis stanu.