Робота з файлами. Менеджер контексту

Файл

- Файл - це іменована область даних на носію інформації.

Файл в Python

- З точки зору Python, файли це спеціальний тип даних який дає інтерфейс доступу та роботи з файлами операційної системи.

- Щоб отримати доступ до файлу ОС, ми повинні "відкрити" даний файл використовуючи метод open()

Метод open(), робота з файлом

- Метод open() очікує отримання жвох аргументів:

1. Обов'язковий - шлях до файлу, і назва файлу. (якщо файл знаходиться в папці з скриптом, то тільки назва файла)
2. Опціональний - модифікатор доступу. За замовчуванням, файл відкривається для читання

Введення виведення даних в Python

- Text IO
- Binary IO (або буферизований)
- Raw IO (небуферизований)

Модифікації доступу до файлів

- "r" ("rt") - (reade) Читання
- "w" ("wt") - (write) запис
- "a" ("at") - (append) дозапис
- "x" - створює файл якщо його не існує, інакше виключення
- "rb" - буферизоване читання
- "wb" - буферизований запис
- "ab" - буферизований дозапис
- "+" - для відкриття одночасно на запис і читання (Приклад "r+")

Метод close()

- Даний метод використовується для закриття файлу. Закривати файл потрібно обов'язково, особливо якщо модифікуємо його вміст. Причина цього в тому що до тих пір поки файл не закрили, модифікований вміст знаходиться в оперативній пам'яті. А метод close() "виштовхує" зміни в постійну пам'ять

Методи для роботи з файлом

- write() записує в файл
- read() - читає весь файл
- readline() - повертає вміст наступного рядка
- readlines() - повертає список всіх рядків

Менеджер контексту і open

```
with open("test.txt", "w") as file:
    file.write("Hello")
```

Використання менеджеру контексту для роботи з файлом, позбавляє нас відповідальності за закриття файлу. Надалі закриттям файлу займається менеджер контексту

Що таке контекстний менеджер

- Менеджер контексту це об'єкт класу, який має реалізованим як мінімум два основних методи
  __enter__(), __exit__()

Реалізація методу open()

```
class File(object):
    def __init__(self, file_name, method):
        self.file_obj = open(file_name, method)
    def __enter__(self):
        return self.file_obj
    def __exit__(self, type, value, traceback):
        self.file_obj.close()

with File('demo.txt', 'w') as opened_file:
    opened_file.write('Hola!')
```

Як це працює

- Метод __exit__ приймає три аргументи. Вони обов'язкові для будь-якого методу __exit__ класу контекстного менеджера. Давайте обговоримо логіку роботи:

1. with зберігає метод __exit__ класу File.
2. Після чого відбувається виклик методу __enter__ класу File.
3. Метод __enter__ відкриває файл і повертає його.
4. Дескриптор файлу передається в opened_file.
5. Ми записуємо інформацію в файл за допомогою .write ().
6. with викликає збережений __exit__ метод.
7. Метод __exit__ закриває файл.

type, value, traceback

- Між четвертим і шостим кроком при виникненні виключення, Python передає тип, значення і зворотний трасування виключення методу __exit__. Це дозволяє методу __exit__ вибирати спосіб закриття файлу і виконувати додаткові дії при необхідності.

- Кроки, які виконує with при виникненні виключення:

1. Тип, значення і зворотна трасування помилки передається в метод __exit__.
2. Обробка виключення передається методу __exit__
3. Якщо __exit__ повертає True, то виняток було коректно оброблено.
4. При поверненні будь-якого іншого значення with викликає виняток.

- Виключення вважається опрацьованим як метод __exit__ повернув True