Context Managers در پایتون

یک **Context Manager** ابزاری است که به‌طور خاص برای مدیریت منابع (مثل فایل‌ها، اتصالات پایگاه داده، و غیره) طراحی شده است. Context Managerها اجازه می‌دهند که منابعی که نیاز به مدیریت دارند به‌طور خودکار باز و بسته شوند، و این کار با استفاده از دستور `with` انجام می‌شود. این الگو باعث می‌شود که کد تمیزتر، خواناتر و مدیریت خطاها ساده‌تر شود.

---

1. استفاده از Context Manager برای مدیریت منابع

در بسیاری از موارد، هنگام کار با منابع خارجی مثل فایل‌ها یا پایگاه‌های داده، نیاز به بستن و آزادسازی منابع پس از استفاده داریم. استفاده از `with` و Context Manager این کار را به‌صورت خودکار انجام می‌دهد و دیگر نیازی به نوشتن کد برای بستن منابع یا مدیریت خطاها به‌صورت دستی نیست.

مثال 1: استفاده از Context Manager برای فایل‌ها

در پایتون برای کار با فایل‌ها معمولاً از `with` استفاده می‌شود تا به‌طور خودکار فایل‌ها باز و بسته شوند. این کار باعث می‌شود که حتی در صورت بروز خطا نیز فایل بسته شود.

```
# استفاده از Context Manager برای باز و بسته کردن فایل‌ها
with open('example.txt', 'r') as file:
    content = file.read()
    print(content)

# پس از این بلاک، فایل به‌طور خودکار بسته می‌شود.
```

در اینجا، فایل به‌طور خودکار پس از انجام عملیات `read` بسته می‌شود، بدون نیاز به استفاده از `file.close()`.

مثال 2: استفاده از Context Manager برای اتصالات پایگاه داده

زمانی که با پایگاه‌های داده کار می‌کنید، مدیریت اتصال و قطع اتصال بسیار مهم است. از یک Context Manager می‌توان برای مدیریت این اتصالات استفاده کرد.

```
import sqlite3

class DatabaseConnection:
    def __init__(self, db_name):
        self.db_name = db_name
        self.connection = None

    def __enter__(self):
        self.connection = sqlite3.connect(self.db_name)
        return self.connection

    def __exit__(self, exc_type, exc_value, traceback):
        if self.connection:
            self.connection.close()

# استفاده از Context Manager برای اتصال به پایگاه داده
with DatabaseConnection('example.db') as conn:
    cursor = conn.cursor()
    cursor.execute('SELECT * FROM users')
    print(cursor.fetchall())
```

```
# در اینجا، اتصال به پایگاه داده به‌طور خودکار بسته می‌شود.
```

در اینجا، با استفاده از `with` و یک Context Manager سفارشی برای اتصال به پایگاه داده، اتصال به‌طور خودکار بسته می‌شود، حتی در صورت بروز خطا.

---

2. تعریف Context Manager سفارشی با استفاده از `with` و متدهای `__enter__` و `__exit__`

برای تعریف یک Context Manager سفارشی در پایتون، باید متدهای `__enter__` و `__exit__` را پیاده‌سازی کنیم:

- `__enter__`: این متد زمانی که `with` وارد بلاک می‌شود، اجرا می‌شود و به‌طور معمول یک شیء یا منبع را باز می‌کند و آن را به تابع `with` می‌دهد.
- `__exit__`: این متد زمانی که اجرای کد داخل بلاک `with` تمام می‌شود، اجرا می‌شود. در این متد معمولاً منابع آزاد یا بسته می‌شوند.

مثال 1: یک Context Manager سفارشی برای فایل

در اینجا یک مثال از ایجاد یک Context Manager سفارشی برای مدیریت فایل‌ها آورده شده است:

```
class FileHandler:
    def __init__(self, filename, mode):
        self.filename = filename
        self.mode = mode
        self.file = None

    def __enter__(self):
        self.file = open(self.filename, self.mode)
        return self.file

    def __exit__(self, exc_type, exc_value, traceback):
        if self.file:
            self.file.close()
            if exc_type:
                print(f"خطا: {exc_value}")

# استفاده از Context Manager سفارشی برای کار با فایل
with FileHandler('example.txt', 'w') as f:
    f.write("Hello, Context Manager!")

# فایل به‌طور خودکار بسته می‌شود.
```

در این مثال، `FileHandler` یک Context Manager سفارشی است که برای باز و بسته کردن فایل‌ها استفاده می‌شود. اگر خطا در داخل بلاک `with` رخ دهد، آن را مدیریت می‌کند و فایل را بسته می‌کند.

مثال 2: مدیریت تراکنش‌ها در پایگاه داده

در اینجا یک مثال دیگر از Context Manager برای مدیریت تراکنش‌ها در پایگاه داده آورده شده است:

```
class Transaction:
    def __init__(self, db_connection):
```

```
        self.db_connection = db_connection
        self.transaction_started = False

    def __enter__(self):
        self.db_connection.begin()
        self.transaction_started = True
        return self.db_connection

    def __exit__(self, exc_type, exc_value, traceback):
        if exc_type is None:  # در صورتی که خطا نباشد، تراکنش را کامیت می‌کنیم
            self.db_connection.commit()
        else:
            self.db_connection.rollback()  # در صورت خطا، تراکنش را لغو می‌کنیم

# استفاده از Context Manager برای مدیریت تراکنش
with Transaction(db_connection) as db:
    db.execute("INSERT INTO users (name, age) VALUES ('Alice', 30)")

# تراکنش به‌طور خودکار commit یا rollback می‌شود.
```

در این مثال، `Transaction` یک Context Manager است که برای شروع و مدیریت تراکنش‌ها در پایگاه داده استفاده می‌شود. اگر خطایی در داخل بلاک `with` رخ دهد، تراکنش لغو می‌شود، در غیر این صورت تراکنش commit می‌شود.

---

3. مزایای استفاده از Context Manager

- **مدیریت خودکار منابع**: Context Managerها به‌طور خودکار منابع را باز و بسته می‌کنند و دیگر نیازی به نوشتن کد اضافی برای بستن منابع نیست.
- **مدیریت خطاها**: در صورت بروز خطا، Context Manager می‌تواند منابع را به‌طور مناسب مدیریت کرده و از نشت منابع جلوگیری کند.
- **کد تمیزتر و خواناتر**: با استفاده از `with` و Context Manager، کد شما تمیزتر و خواناتر می‌شود و مدیریت منابع به‌راحتی انجام می‌شود.

نتیجه‌گیری

Context Managerها ابزاری قدرتمند برای مدیریت منابع و انجام عملیات‌های قبل و بعد از اجرای یک بخش از کد هستند. این ابزار کمک می‌کند که کدهای شما ساده‌تر، ایمن‌تر و مقیاس‌پذیرتر باشند.