

Kurs rozszerzony języka Python

Wykład 15.

Marcin Młotkowski

1 lutego 2023

Plan wykładu

- 1 Dekoratory
- 2 Manager kontekstu
 - Przykład

Plan wykładu

- 1 Dekoratory
- 2 Manager kontekstu
 - Przykład

Rozszerzanie właściwości funkcji

```
def szalenie_skomplikowana_funkcja(arg1, arg2, arg3):  
    ...
```

Śledzenie wywołania funkcji

Chcemy śledzić wywołania zaimplementowanych funkcji, tj. informacje o wywołaniu oraz informacja o argumentach wywołania. Bez ingerowania w te funkcje.

Schemat rozwiązania

Rozwiązanie 1.

```
def log_foo(*args):  
    print(f"Wywoływana funkcja: foo z argumentami {args}")  
    return foo(*args)
```

Schemat rozwiązania

Rozwiązanie 1.

```
def log_foo(*args):  
    print(f"Wywoływana funkcja: foo z argumentami {args}")  
    return foo(*args)
```

Co z tym zrobić 1.

Zamiast *foo* używamy *log.foo*.

Schemat rozwiązania

Rozwiązanie 1.

```
def log_foo(*args):  
    print(f"Wywoływana funkcja: foo z argumentami {args}")  
    return foo(*args)
```

Co z tym zrobić 1.

Zamiast *foo* używamy *log.foo*.

Co z tym zrobić 2.

```
foo = log.foo
```

Uniwersalna funkcja opakująca inne funkcje

```
def log(fun):  
    def opakowanie(*args):  
        print("funkcja:", fun.__name__, "argumenty:", args)  
        return fun(*args)  
    return opakowanie
```


Uniwersalna funkcja opakująca inne funkcje

```
def log(fun):  
    def opakowanie(*args):  
        print("funkcja:", fun.__name__, "argumenty:", args)  
        return fun(*args)  
    return opakowanie
```

Zastosowanie

```
foo = log(foo)
```

Dekoratory

```
def log(fun):  
    def opakowanie(*args):  
        print(f"{fun.__name__} {args}")  
        return fun(*args)  
    return opakowanie
```

Dekoratory

```
def log(fun):  
    def opakowanie(*args):  
        print(f"{fun.__name__} {args}")  
        return fun(*args)  
    return opakowanie
```

Zastosowanie

```
@log  
def fib(args):  
    ...
```

Pomiar czasu wykonania funkcji

```
def wydajnosc(fun):  
    def opakowanie(*args):  
        t_start = time.perf_counter()  
        wynik = fun(*args)  
        czas = time.perf_counter() - t_start  
        print(f"{fun.__name__} {args} {czas}")  
        return wynik  
    return opakowanie
```

Przykład z życia: keshowanie wyników

```
def cache_function(func):  
    """Keshowanie zapytań"""  
    def wrapper(conn, query, typ):  
        # inicjowanie cache  
        db = shelve.open("/tmp/query.cache")  
        if query in db:  
            res = db[query]  
            return res  
        res = func(conn, query, typ)  
        db[query] = list(res)  
        db.sync()  
        res = db[query]  
        return res  
    return wrapper
```

Dekoratory standardowe

Dekorowanie programów wielowątkowych

```
from threading import Lock
```

```
my_lock = Lock()
```

```
@synchronized(my_lock)  
def critical1(): ...
```

```
@synchronized(my_lock)  
def critical2(): ...
```

Implementacja dekoratora

```
def synchronized(lock):  
    def wrap(f):  
        def new_function(*args, **kw):  
            lock.acquire()  
            try:  
                return f(*args, **kw)  
            finally:  
                lock.release()  
        return new_function  
    return wrap
```

Plan wykładu

- 1 Dekoratory
- 2 Manager kontekstu
 - Przykład

Instrukcja **with-as**

```
with open("file.txt", 'r') as fh:  
    print(fh.read())
```

jest równoważne

```
try:  
    fh = open("file.txt", 'r')  
    print(fh.read())  
except:  
    # obsługa błędu  
finally:  
    fh.close()
```

context manager

Context manager to obiekt implementujący dwie metody:

```
__enter__(self) i  
__exit__(self, exc_type, exc_val, exc_tb)
```

```
open("file.txt", 'r')
```

Funkcja `open()` nie tylko zwraca uchwyt do pliku, ale także zarządzanie kontekstem.

context manager

Context manager to obiekt implementujący dwie metody:

```
__enter__(self) i  
__exit__(self, exc_type, exc_val, exc_tb)
```

```
open("file.txt", 'r')
```

Funkcja `open()` nie tylko zwraca uchwyt do pliku, ale także zarządzanie kontekstem.

```
help(open("plik.txt", 'w'))
```

Podsumowanie

Context manager implementuje szkielet zarządzania kontekstem, instrukcja `with-as` zarządza tym kontekstem.

Przykład: pomiar czasu grupy instrukcji

```
with Pomiar("Blok opomiarowany"):  
    fib(n)
```

Przykład: pomiar czasu grupy instrukcji

```
with Pomiar("Blok opomiarowany"):  
    fib(n)
```

Implementacja

```
class Pomiar:
    def __init__(self, nazwa):
        self.nazwa = nazwa

    def __enter__(self):
        self.t = time.perf_counter()
        return self

    def __exit__(self, exc_type, exc_val, exc_tb):
        czas = time.perf_counter() - self.t
        print(f"Pomiar bloku {self.nazwa} {czas:0.2f}")
        return True
```

Wasze programy :)

```
engine = create_engine('sqlite:///wyklad.db', echo=True)
Base.metadata.create_all(engine)
Session = sessionmaker(bind=engine)
```

```
@app.route
def dodaj(osoba):
    sesja = Session()
    sesja.add(osoba)
    sesja.commit()
```

```
@app.route
def usun(osoba):
    sesja = Session()
    sesja.delete(osoba)
    sesja.commit()
```


Zadanie

Chcę mieć obsługę żądań zapakowaną w kontekście i jeszcze mieć logowanie błędów.

Definiujemy klasę

```
import logging
logging.basicConfig(format='%(asctime)s %(message)s', file=

class AlchSession:
    def __enter__(self):
        logging.info("Początek sesji alchemy")
        self.session = Session()
        return session

    def __exit__(self, exc_type, exc_val, exc_tb):
        logging.info("Koniec sesji alchemy")
        if exc_val is not None:
            logging.info(f"{exc_type}")
            self.session.rollback()
        return True
```

Nowa implementacja

Stary kod

```
@app.route
def dodaj(osoba):
    sesja = Session()
    sesja.add(osoba)
    sesja.commit()
```

zamieniamy na

```
@app.route
def dodaj(osoba):
    with AlchSession() as session:
        sesja.add(osoba)
```

Context manager + generator + dekoratory

Menadżery kontekstu można tworzyć za pomocą dekoratorów.

Implementacja

```
from contextlib import contextmanager

@contextmanager
def session_scope():
    session = Session()
    try:
        yield session
        session.commit()
    except:
        session.rollback()
        raise
    finally:
        session.close()
```

Wykorzystanie

```
with session_scope() as session:  
    session.add(obiekt)
```

Plan wykładu

- 1 Dekoratory
- 2 Manager kontekstu
 - Przykład

