Отчет

Лабораторная работа 7.

Логирование и обработка ошибок в Python

Выполнил: Орлов Михаил
Преподаватель: Жуков Николай Николаевич

Цель работы

- освоить принципы разработки декораторов с параметрами;
- научиться разделять ответственность функций (бизнес-логика) и декораторов (сквозная логика);
- научиться обрабатывать исключения, возникающие при работе с внешними API;
- освоить логирование в разные типы потоков (sys.stdout, io.StringIO, logging);
- научиться тестировать функцию и поведение логирования.

Исходный код декоратора с параметрами

logtools.py > trace

```python
import sys
import logging
import functools
from typing import Any, Callable


def trace(func: Callable = None, *, handle=sys.stdout):
    """
    Универсальный логирующий декоратор.
    Поддерживает:
        • обычные потоки (stdout, StringIO) через write()
        • logging.Logger через info()/error()

    Параметры:
        func   — функция, которую оборачиваем
        handle — поток или логгер
    """

    # Проверяем, что передан логгер (у него есть .info() и нет .write())
    def _is_logger(target) -> bool:
        return isinstance(target, logging.Logger)

    is_logger = _is_logger(handle)

    # Функции записи логов в поток или логгер
    def log_info(message: str) -> None:
        if is_logger:
            handle.info(message)
        else:
            handle.write(f"INFO: {message}\n")

    def log_error(message: str) -> None:
        if is_logger:
            handle.error(message)
        else:
            handle.write(f"ERROR: {message}\n")

    def decorator(fn: Callable) -> Callable:
        @functools.wraps(fn)
        def wrapped(*args, **kwargs) -> Any:
            # Формирование сигнатуры вызова
            args_list = [repr(a) for a in args]
            kwargs_list = [f"{k}={repr(v)}" for k, v in kwargs.items()]
            call_repr = ", ".join(args_list + kwargs_list)

            # Логируем старт
            log_info(f"Запуск {fn.__name__}({call_repr})")

```

```python
            try:
                result = fn(*args, **kwargs)
                log_info(f"{fn.__name__} вернула {repr(result)}")
                return result
            except Exception as exc:  # Логируем любую ошибку
                exc_type = type(exc).__name__
                log_error(f"Ошибка в {fn.__name__}: {exc_type}: {exc}")
                raise

        return wrapped

    # Декоратор вызван как @trace(handle=…)
    if func is None:
        return decorator

    # Декоратор вызван как @trace
    return decorator(func)


# Дополнительно создадим логгер для записи в файл
file_logger = logging.getLogger("currency_file")
file_handler = logging.FileHandler("currency.log", encoding="utf-8")
file_formatter = logging.Formatter("%(levelname)s: %(message)s")
file_handler.setFormatter(file_formatter)
file_logger.addHandler(file_handler)
file_logger.setLevel(logging.INFO)

```

Исходный код `get_currencies` (без логирования)

get_currencies_demo.py > ...

```python
import requests

def get_currencies(codes):
    """
    Получает курсы валют по списку кодов из API ЦБ РФ.
    Логирования нет — функция чистая и возвращает только данные.
    """

    url = "https://www.cbr-xml-daily.ru/latest.js"
    data = requests.get(url).json()
    rates = data.get("rates", {})

    result = {}
    for code in codes:
        result[code] = rates.get(code)

    return result
```

Демонстрационный пример (квадратное уравнение)

```
@trace(handle=quad_logger)
def demo_quad(a, b, c):
    return solve_quadratic(a, b, c)


print("→ Два корня:", demo_quad(1, -5, 6))
print("→ Один корень:", demo_quad(1, -4, 4))

try:
    print("→ Нет корней:", demo_quad(1, 0, 1))
except Exception as e:
    print("→ Ошибка: нет корней:", e)

try:
    demo_quad(0, 2, 3)
except Exception as e:
    print("→ Ошибка a=0:", e)

try:
    demo_quad("abc", 2, 3)
except Exception as e:
    print("→ Ошибка: неправильный тип:", e)



print("\nГотово. Все логи записаны в currency.log и quad.log")
```

Скриншоты / фрагменты логов

```
INFO: Запуск demo_quad(1, -5, 6)
INFO: demo_quad вернула (3.0, 2.0)
INFO: Запуск demo_quad(1, -4, 4)
INFO: demo_quad вернула (2.0, 2.0)
INFO: Запуск demo_quad(1, 0, 1)
ERROR: Ошибка в demo_quad: ValueError: Дискриминант меньше нуля — корней нет
INFO: Запуск demo_quad(1, -5, 6)
INFO: demo_quad вернула (3.0, 2.0)
INFO: Запуск demo_quad(1, -4, 4)
INFO: demo_quad вернула (2.0, 2.0)
INFO: Запуск demo_quad(1, 0, 1)
ERROR: Ошибка в demo_quad: ValueError: Дискриминант меньше нуля — корней нет
INFO: Запуск demo_quad(1, -5, 6)
INFO: demo_quad вернула (3.0, 2.0)
INFO: Запуск demo_quad(1, -4, 4)
INFO: demo_quad вернула (2.0, 2.0)
INFO: Запуск demo_quad(1, 0, 1)
ERROR: Ошибка в demo_quad: ValueError: Дискриминант меньше нуля — корней нет
INFO: Запуск demo_quad(0, 2, 3)
ERROR: Ошибка в demo_quad: ValueError: Коэффициент 'a' не может быть равен нулю в квадратном уравнении
INFO: Запуск demo_quad('abc', 2, 3)
ERROR: Ошибка в demo_quad: TypeError: Коэффициенты должны быть числами
```

currency.log

```
INFO: Запуск get_currencies(['USD'])
INFO: get_currencies вернула {'USD': 76.0937}
INFO: Запуск demo_usd_eur()
INFO: Запуск get_currencies(['USD', 'EUR'])
INFO: get_currencies вернула {'USD': 76.0937, 'EUR': 88.7028}
INFO: demo_usd_eur вернула {'USD': 76.0937, 'EUR': 88.7028}
INFO: Запуск get_currencies(['USD'])
INFO: get_currencies вернула {'USD': 76.0937}
INFO: Запуск demo_usd_eur()
INFO: Запуск get_currencies(['USD', 'EUR'])
INFO: get_currencies вернула {'USD': 76.0937, 'EUR': 88.7028}
INFO: demo_usd_eur вернула {'USD': 76.0937, 'EUR': 88.7028}
INFO: Запуск solve_quadratic(1, -5, 6)
INFO: solve_quadratic вернула (3.0, 2.0)
INFO: Запуск solve_quadratic(1, -4, 4)
INFO: solve_quadratic вернула (2.0, 2.0)
```

Тесты

```python
class TestTraceDecorator(unittest.TestCase):

    def test_trace_success(self):
        """Проверка корректного логирования успешного вызова."""
        log = StringIO()

        @trace(handle=log)
        def add(a, b):
            return a + b

        result = add(2, 3)
        log_content = log.getvalue()

        self.assertEqual(result, 5)
        self.assertIn("INFO: Запуск add(2, 3)", log_content)
        self.assertIn("INFO: add вернула 5", log_content)

    def test_trace_exception(self):
        """Проверка логирования ошибки."""
        log = StringIO()

        @trace(handle=log)
        def bad(x):
            raise ValueError("ошибка!")

        with self.assertRaises(ValueError):
            bad(10)

        log_content = log.getvalue()

        self.assertIn("INFO: Запуск bad(10)", log_content)
        self.assertIn("ERROR: Ошибка в bad: ValueError: ошибка!", log_content)
```

```python
class TestStringIOLogging(unittest.TestCase):

    def test_logging_stringio(self):
        """Проверка записи в StringIO."""
        log = StringIO()

        @trace(handle=log)
        def fake():
            return {"USD": 100}

        res = fake()
        log_content = log.getvalue()

        self.assertEqual(res, {"USD": 100})
        self.assertIn("INFO: Запуск fake()", log_content)
        self.assertIn("INFO: fake вернула {'USD': 100}", log_content)
```

```python
class TestQuadraticSolver(unittest.TestCase):

    def test_two_roots(self):
        x1, x2 = solve_quadratic(1, -3, 2)
        self.assertEqual((x1, x2), (2.0, 1.0))

    def test_one_root(self):
        x1, x2 = solve_quadratic(1, 2, 1)
        self.assertEqual((x1, x2), (-1.0, -1.0))

    def test_negative_discriminant(self):
        with self.assertRaises(ValueError):
            solve_quadratic(1, 0, 1)

    def test_wrong_type(self):
        with self.assertRaises(TypeError):
            solve_quadratic("abc", 1, 1)
```