Лабораторная работа 7. Логирование и обработка ошибок в Python

Отчёт

Выполнил Жохов Даниил (502506)
Группа P3123

Исходный код декоратора:

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

def logger(func=None, *, handle=sys.stdout):
    """
    Параметризуемый декоратор для логирования вызовов функции.
    Поддерживает:
        - sys.stdout / sys.stderr (или любой file-like объект с .write())
        - io.StringIO / другие file-like объекты
        - logging.Logger — тогда использует .info() и .error()
    """
    def decorator(func: Callable) -> Callable:
        @functools.wraps(func)
        def wrapper(*args, **kwargs) -> Any:
            use_logging = isinstance(handle, logging.Logger)

            # Форматирование аргументов
            args_repr = [repr(a) for a in args]
            kwargs_repr = [f"{k}={repr(v)}" for k, v in kwargs.items()]
            signature = ", ".join(args_repr + kwargs_repr)

            if use_logging:
                handle.info(f"Вызов функции {func.__name__} с аргументами: ({signature})")
            else:
                handle.write(f"INFO: Вызов функции {func.__name__} с аргументами: ({signature})\n")

            try:
                result = func(*args, **kwargs)
                if use_logging:
                    handle.info(f"Функция {func.__name__} успешно завершена. Результат: {repr(result)}")
                else:
                    handle.write(f"INFO: Функция {func.__name__} успешно завершена. Результат: {repr(result)}\n")
                return result
```

```
            except Exception as e:
                if use_logging:
                    handle.error(f"Исключение в функции {func.__name__}: {type(e).__name__}: {e}")
                else:
                    handle.write(f"ERROR: Исключение в функции {func.__name__}: {type(e).__name__}: {e}\n")
                raise
        return wrapper

    if func is None:
        return decorator
    else:
        return decorator(func)
```

Исходный код get_currency

```
import requests
from logger import logger

def get_currencies(currency_codes: list, url="https://www.cbr-xml-daily.ru/daily_json.js") -> dict:
    """
    Получает курсы валют с API ЦБ РФ.
    Возвращает словарь вида: {"USD": 93.25, "EUR": 101.7}
    """
    try:
        response = requests.get(url)
        response.raise_for_status()
    except requests.RequestException as e:
        raise ConnectionError(f"Не удаётся подключиться к API: {e}")

    try:
        data = response.json()
    except ValueError as e:
        raise ValueError(f"Некорректный JSON: {e}")

    if "Valute" not in data:
        raise KeyError("Ответ не содержит ключа 'Valute'")

    valute = data["Valute"]
    result = {}
    for code in currency_codes:
        if code not in valute:
            raise KeyError(f"Валюта {code} отсутствует в данных")
        currency_info = valute[code]
        if "Value" not in currency_info:
            raise KeyError(f"Для валюты {code} отсутствует значение 'Value'")
        value = currency_info["Value"]
        if not isinstance(value, (int, float)):
            raise TypeError(f"Курс валюты {code} имеет неверный тип: {type(value)}")
        result[code] = float(value)
    return result
```

Демонстрационный пример

```python
import logging
import math

# Настройка логирования в файл quadratic.log
logging.basicConfig(
    filename="quadratic.log",
    level=logging.DEBUG,
    format="%(levelname)s: %(message)s"
)

def solve_quadratic(a, b, c):
    logging.info(f"Solving equation: {a}x^2 + {b}x + {c} = 0")
    for name, value in zip(("a", "b", "c"), (a, b, c)):
        if not isinstance(value, (int, float)):
            logging.critical(f"Parameter '{name}' must be a number, got: {value}")
            raise TypeError(f"Coefficient '{name}' must be numeric")

    if a == 0:
        logging.error("Coefficient 'a' cannot be zero")
        raise ValueError("a cannot be zero")

    d = b * b - 4 * a * c
    logging.debug(f"Discriminant: {d}")

    if d < 0:
        logging.warning("Discriminant < 0: no real roots")
        return None

    if d == 0:
        x = -b / (2 * a)
        logging.info("One real root")
        return (x,)

    root1 = (-b + math.sqrt(d)) / (2 * a)
    root2 = (-b - math.sqrt(d)) / (2 * a)
    logging.info("Two real roots computed")
    return root1, root2
```

```python

# Демонстрационные вызовы
if __name__ == "__main__":
    print("=== Демонстрация solve_quadratic с логированием ===")

    # Успешные случаи
    print("→ Два корня:", solve_quadratic(1, -5, 6))      # (3.0, 2.0)
    print("→ Один корень:", solve_quadratic(1, -4, 4))    # (2.0,)

    # Предупреждение: нет вещественных корней
    print("→ Нет вещественных корней:", solve_quadratic(1, 0, 1))  # None

    # Ошибка: a = 0
    try:
        solve_quadratic(0, 2, 3)
    except ValueError as e:
        print("→ Ошибка (a=0):", e)

    # Критическая ошибка: некорректный тип
    try:
        solve_quadratic("abc", 2, 3)
    except TypeError as e:
        print("→ Критическая ошибка (тип):", e)
```

Логи

```
INFO: Solving equation: 1x^2 + -5x + 6 = 0
DEBUG: Discriminant: 1
INFO: Two real roots computed
INFO: Solving equation: 1x^2 + -4x + 4 = 0
DEBUG: Discriminant: 0
INFO: One real root
INFO: Solving equation: 1x^2 + 0x + 1 = 0
DEBUG: Discriminant: -4
WARNING: Discriminant < 0: no real roots
INFO: Solving equation: 0x^2 + 2x + 3 = 0
ERROR: Coefficient 'a' cannot be zero
INFO: Solving equation: abcx^2 + 2x + 3 = 0
CRITICAL: Parameter 'a' must be a number, got: abc
```

```
=== Логирование в stdout ===
INFO: Вызов функции get_usd с аргументами: ()
INFO: Функция get_usd успешно завершена. Результат: {'USD': 76.9708}
Результат: {'USD': 76.9708}

=== Логирование в StringIO ===
Результат: {'EUR': 89.9011}
Лог из StringIO:
INFO: Вызов функции get_eur с аргументами: ()
INFO: Функция get_eur успешно завершена. Результат: {'EUR': 89.9011}


=== Логирование в файл currency.log ===
Результат: {'USD': 76.9708, 'EUR': 89.9011}

 Все логи записаны в файлы.
```

Тесты

```python
import unittest
import io
from unittest.mock import patch, Mock
import requests  # ← добавьте импорт requests!
from get_currency import get_currencies
from logger import logger


class TestGetCurrencies(unittest.TestCase):

    @patch("get_currency.requests.get")
    def test_valid_response(self, mock_get):
        mock_response = Mock()
        mock_response.json.return_value = {
            "Valute": {
                "USD": {"Value": 93.25},
                "EUR": {"Value": 101.7}
            }
        }
        mock_response.raise_for_status.return_value = None
        mock_get.return_value = mock_response

        result = get_currencies(["USD", "EUR"])
        self.assertEqual(result, {"USD": 93.25, "EUR": 101.7})

    @patch("get_currency.requests.get")
    def test_connection_error(self, mock_get):
        # ✅ Используем настоящий requests.RequestException
        mock_get.side_effect = requests.RequestException("Network error")
        with self.assertRaises(ConnectionError):
            get_currencies(["USD"])

    @patch("get_currency.requests.get")
    def test_invalid_json(self, mock_get):
        mock_response = Mock()
        mock_response.raise_for_status.return_value = None
        mock_response.json.side_effect = ValueError("Invalid JSON")
```

```
38              mock_get.return_value = mock_response
39              with self.assertRaises(ValueError):
40                  get_currencies(["USD"])
41
42          @patch("get_currency.requests.get")
43          def test_missing_valute_key(self, mock_get):
44              mock_response = Mock()
45              mock_response.json.return_value = {"Other": "data"}
46              mock_response.raise_for_status.return_value = None
47              mock_get.return_value = mock_response
48              with self.assertRaises(KeyError):
49                  get_currencies(["USD"])
50
51          @patch("get_currency.requests.get")
52          def test_currency_not_found(self, mock_get):
53              mock_response = Mock()
54              mock_response.json.return_value = {"Valute": {"EUR": {"Value": 100}}}
55              mock_response.raise_for_status.return_value = None
56              mock_get.return_value = mock_response
57              with self.assertRaises(KeyError):
58                  get_currencies(["USD"])
59
60          @patch("get_currency.requests.get")
61          def test_invalid_currency_type(self, mock_get):
62              mock_response = Mock()
63              mock_response.json.return_value = {"Valute": {"USD": {"Value": "ninety"}}}
64              mock_response.raise_for_status.return_value = None
65              mock_get.return_value = mock_response
66              with self.assertRaises(TypeError):
67                  get_currencies(["USD"])
68
69
70  class TestLoggerDecorator(unittest.TestCase):
```

```
71
72      def test_successful_execution(self):
73          stream = io.StringIO()
74
75          @logger(handle=stream)
76          def multiply(x, y):
77              return x * y
78
79          result = multiply(3, 4)
80          self.assertEqual(result, 12)
81
82          logs = stream.getvalue()
83          self.assertIn("Вызов функции multiply с аргументами: (3, 4)", logs)
84          self.assertIn("успешно завершена. Результат: 12", logs)
85
86      def test_exception_raised(self):
87          stream = io.StringIO()
88
89          @logger(handle=stream)
90          def divide(x, y):
91              return x / y
92
93          with self.assertRaises(ZeroDivisionError):
94              divide(10, 0)
95
96          logs = stream.getvalue()
97          self.assertIn("Исключение в функции divide: ZeroDivisionError", logs)
98
99
100 class TestStreamWrite(unittest.TestCase):
101
102     def setUp(self):
103         self.stream = io.StringIO()
```

```
            self.stream = io.StringIO()

            @logger(handle=self.stream)
            def wrapped():
                return get_currencies(['USD'], url="https://invalid")

            self.wrapped = wrapped

        @patch("get_currency.requests.get")
        def test_logging_error(self, mock_get):
            mock_get.side_effect = requests.RequestException("Invalid URL")
            with self.assertRaises(ConnectionError):
                self.wrapped()

            logs = self.stream.getvalue()
            self.assertIn("ERROR", logs)
            self.assertIn("ConnectionError", logs)


    if __name__ == '__main__':
        unittest.main()
```

Все тесты успешно пройдены.