Министерство науки и высшего образования Российской Федерации

федеральное государственное автономное образовательное учреждение высшего образования

“НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО”

Факультет	**Программной Инженерии и Компьютерной Техники**
Направление подготовки (специальность)	**Нейротехнологии и программирование**
Дисциплина	**Программирование на Python**

Лабораторная работа 7
ОТЧЕТ

Выполнил студент:	**Иголкин Владислав Андреевич (504623)**
Группа:	**P3124**
Преподаватель:	**Жуков Николай Николаевич (261087)**

г. Санкт-Петербург

2025 г.

Цели 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.

1. Исходный код декоратора с параметрами:

```
# 1. Декоратор logger
""" Универсальный логирующий декоратор.

    Логирует старт, завершение и ошибки функции.
    Поддерживает логирование в sys.stdout или logging.Logger.

    Args:
        func (callable): Функция для оборачивания.
        handle (sys.stdout или logging.Logger): Куда писать логи.
    """
def logger(func=None, *, handle=sys.stdout):
    if func is None:
        return lambda f: logger(f, handle=handle)

    is_logger = isinstance(handle, logging.Logger)

    @functools.wraps(func)
    def wrapper(*args, **kwargs):
        msg_start = f"START {func.__name__} args={args}, kwargs={kwargs}"
        if is_logger:
            handle.info(msg_start)
        else:
            handle.write(msg_start + "\n")

        try:
            result = func(*args, **kwargs)
        except Exception as e:
            msg_err = f"ERROR in {func.__name__}: {type(e).__name__}: {e}"
            if is_logger:
                handle.error(msg_err)
            else:
                handle.write(msg_err + "\n")
            raise

        msg_end = f"END {func.__name__} result={result}"
        if is_logger:
            handle.info(msg_end)
        else:
            handle.write(msg_end + "\n")

        return result

    return wrapper
```

2. Исходный код get_currencies (без логирования):

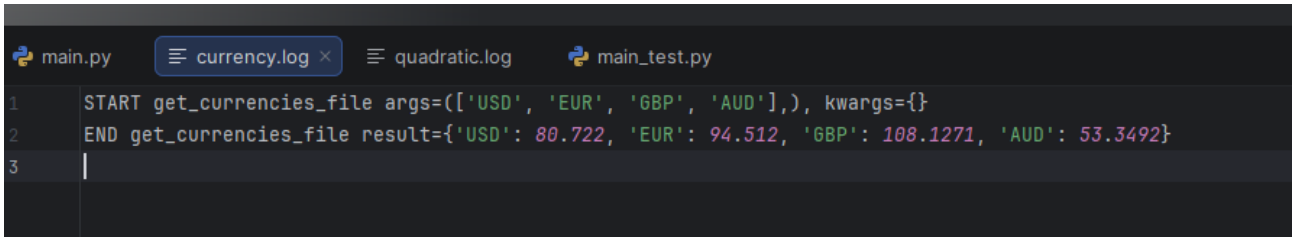
```python
# 2. Функция get_currencies
def get_currencies(currency_codes: list,url: str = "https://www.cbr-xml-daily.ru/daily_json.js") -> dict:
    """Получает курсы валют с API Центробанка России.

    Args:
        currency_codes (List[str]): Список символьных кодов валют (например, ['USD', 'EUR']).
        url (str, optional): URL API. По умолчанию 'https://www.cbr-xml-daily.ru/daily_json.js'.

    Returns:
        Dict[str, float]: Словарь с курсами валют.

    Raises:
        ConnectionError: Если API недоступен.
        ValueError: Если JSON некорректный.
        KeyError: Если отсутствуют ожидаемые ключи в данных.
        TypeError: Если курс валюты имеет неверный тип.
    """
    try:
        response = requests.get(url)
        response.raise_for_status()
    except requests.RequestException as e:
        raise ConnectionError("API недоступен") from e

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

    if "Valute" not in data:
        raise KeyError("Нет ключа 'Valute'")

    result = {}

    for code in currency_codes:
        if code not in data["Valute"]:
            raise KeyError(f"Валюта {code} отсутствует в данных")

        value = data["Valute"][code].get("Value")

        if not isinstance(value, (int, float)):
            raise TypeError(f"Курс валюты {code} имеет неверный тип")

        result[code] = value

    return result
```

3. Демонстрационный пример (квадратное уравнение):

```python
# Функция solve_quadratic
logging.basicConfig(
    filename="quadratic.log",
    level=logging.DEBUG,
    format="%(levelname)s: %(message)s"
)

quad_logger = logging.getLogger("quadratic")

@logger(handle=quad_logger)
def solve_quadratic(a: float, b: float, c: float) -> Optional[Tuple[float, ...]]:
    """Решает квадратное уравнение ax^2 + bx + c = 0 с логированием.
```

```
    Args:
        a (float): Коэффициент при x^2.
        b (float): Коэффициент при x.
        c (float): Свободный член.

    Returns:
        Optional[Tuple[float, ...]]: Кортеж с корнями (1 или 2) или None, если корней нет.

    Raises:
        TypeError: Если коэффициенты некорректного типа.
        ValueError: Если уравнение невозможно (a=b=0).
    """
    # ERROR — некорректные данные
    for name, value in zip(("a", "b", "c"), (a, b, c)):
        if not isinstance(value, (int, float)):
            quad_logger.error(f"Invalid type for '{name}': {value}")
            raise TypeError(f"Coefficient '{name}' must be numeric")

    # CRITICAL — полностью невозможная ситуация
    if a == 0 and b == 0:
        quad_logger.critical("Impossible equation: a = 0 and b = 0")
        raise ValueError("Invalid quadratic equation")

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

    # WARNING — нет действительных корней
    if d < 0:
        quad_logger.warning("Discriminant < 0: no real roots")
        return None

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

    # INFO — два корня
    x1 = (-b + math.sqrt(d)) / (2 * a)
    x2 = (-b - math.sqrt(d)) / (2 * a)
    quad_logger.info("Two real roots computed")
    return x1, x2
```

4. Скриншоты / фрагменты логов:



main.py | currency.log | quadratic.log | main_test.py

```
INFO: START solve_quadratic args=(1, -3, 2), kwargs={}
DEBUG: Discriminant = 1
INFO: Two real roots computed
INFO: END solve_quadratic result=(2.0, 1.0)
INFO: START solve_quadratic args=(1, 1, 10), kwargs={}
DEBUG: Discriminant = -39
WARNING: Discriminant < 0: no real roots
INFO: END solve_quadratic result=None
INFO: START solve_quadratic args=('abc', 2, 3), kwargs={}
ERROR: Invalid type for 'a': abc
ERROR: ERROR in solve_quadratic: TypeError: Coefficient 'a' must be numeric
INFO: START solve_quadratic args=(0, 0, 5), kwargs={}
CRITICAL: Impossible equation: a = 0 and b = 0
ERROR: ERROR in solve_quadratic: ValueError: Invalid quadratic equation

```

5. Тесты:

```python
from main import *
import unittest
import io


class TestGetCurrencies(unittest.TestCase):
    """Тесты для функции get_currencies."""
    def test_real_currencies(self):
        """Проверка возврата корректных курсов валют."""
        result = get_currencies(["USD", "EUR"])
        self.assertIn("USD", result)
        self.assertIn("EUR", result)
        self.assertIsInstance(result["USD"], (int, float))

    def test_missing_currency(self):
        """Проверка ошибки при несуществующей валюте."""
        with self.assertRaises(KeyError):
            get_currencies(["XXX"])

    def test_connection_error(self):
        """Проверка ошибки ConnectionError при недоступном API."""
        with self.assertRaises(ConnectionError):
            get_currencies(["USD"], url="https://invalid")

    def test_invalid_json(self):
        """Проверка ValueError при некорректном JSON."""
        with self.assertRaises(ValueError):
            get_currencies(["USD"], url="https://httpbin.org/html")

class TestLoggerDecorator(unittest.TestCase):
    """Тесты для логирующего декоратора logger."""
    def setUp(self):
        """Создание потока для перехвата логов."""
        self.stream = io.StringIO()

    def test_success_logging(self):
        """Проверка логов при успешном выполнении функции."""
        @logger(handle=self.stream)
```

```
        def test_func(x):
            """Функция для тестирования."""
            return x * 2

        result = test_func(4)
        logs = self.stream.getvalue()

        self.assertEqual(result, 8)
        self.assertIn("START test_func", logs)
        self.assertIn("END test_func result=8", logs)

    def test_error_logging(self):
        """Проверка логов и проброса исключения при ошибке."""
        @logger(handle=self.stream)
        def test_func(x):
            """Функция, вызывающая исключение."""
            raise ValueError("boom")

        with self.assertRaises(ValueError):
            test_func(1)

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

class TestStreamWrite(unittest.TestCase):
    """Пример теста с использованием контекста и StringIO."""
    def setUp(self):
        self.stream = io.StringIO()

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

        self.wrapped = wrapped

    def test_logging_error(self):
        """Проверка логирования ошибки при недоступном API."""
        with self.assertRaises(ConnectionError):
            self.wrapped()

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

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

```
C:\Users\solik\AppData\Local\Programs\Python\Python312\python.exe C:\Users\s
.......
----------------------------------------------------------------------
Ran 7 tests in 6.477s

OK

Process finished with exit code 0
```