Лабораторная работа 1

Задача № 1

Условие: Создайте простое замыкание (closure) в виде внутренней (вложенной) функции внутри обычной функции. Внутренняя функция (замыкание, closure) должна использовать переменные и аргументы обычной функции, в которую она вложена. Внутри внутренней функции (closure) распечатайте переданные аргументы в терминале. Верните вложенную функцию из обычной функции с помощью выражения return.

Код программы и результат

```
def main(*c):
    def func():
        for index, value in enumerate(c):
            print(f"A-{index + 1}: {value}")

    return func

c = main("Привет", "Мир", "Это", "замыкание", "в", "Python")
c()
```

```
A-1: Привет
A-2: Мир
A-3: Это
A-4: замыкание
A-5: в
A-6: Python
```

Как работает код:

Код определяет функцию main, которая принимает произвольные аргументы. Внутри неё создаётся функция func, выводящая каждый аргумент с его индексом. main возвращает func, а вызов c() выводит строки с индексами.

Задание №2

Условие: Изучите на примерах в интернете, что такое closure и и как их применять для создания простого декоратора (decorator) с @-синтаксисом в Python. Модернизируйте калькулятор из задачи 3.1 лабораторной работы №1. Декорируйте вашу функцию calculate. В соответствующем декорирующем замыкании, в closure, то есть во внутренней функции используйте простое логирование (стандартный модуль Python logging). Сделайте логирование внутри замыкания до вызова вашей функции calculate(operand1, operand2, action), в котором логируется информация о том какие операнды и какая арифметическая операция собираются поступить на вход функции calculate(operand1, operand2, action). Затем внутри того же closure следует сам вызов функции calculate(...). А затем, после этого вызова должно быть снова логирование, но уже с результатом выполнения вычисления, проделанного в этой функции.

Код программы и результат:

```
def log_decorator(func):
    def wrapper(num1, num2, operator):
        print(f"Вызов функции: {func.__name__} с аргументами:
{num1}, {num2}, оператор: {operator}")
        result = func(num1, num2, operator)
        print(f"Результат: {result}")
        return result
    return wrapper

@log_decorator
def calculate(num1, num2, operator):
    if operator == "+":
        return num1 + num2
    elif operator == "-":
        return num1 - num2
    elif operator == "*":
        return num1 * num2
    elif operator == "/":
        if num2 == 0:
            return "Ошибочка!"
        else:
            return num1 / num2
    else:
        return "Ошибочка!"

def test_1():
    assert calculate(2, 3, "+") == 5, "ошибочка"

def test_2():
    assert calculate(5, 2, "-") == 3, "ошибочка"

def test_3():
    assert calculate(4, 5, "*") == 20, "ошибочка"

def test_4():
    assert calculate(10, 3, "/") == 3.3333333333333335,
"ошибочка"

def main():
    num1 = float(input("Введите первое число: "))
    num2 = float(input("Введите второе число: "))
    operator = input("Введите оператор: ")
    res = calculate(num1, num2, operator)
    print(res)

test_1()
test_2()
test_3()
test_4()
main()
```

```
Вызов функции: calculate с аргументами: 2, 3, оператор: +
Результат: 5
Вызов функции: calculate с аргументами: 5, 2, оператор: -
Результат: 3
Вызов функции: calculate с аргументами: 4, 5, оператор: *
Результат: 20
Вызов функции: calculate с аргументами: 10, 3, оператор: /
Результат: 3.3333333333333335
Введите первое число: 10
Введите второе число: 20
Введите оператор: *
Вызов функции: calculate с аргументами: 10.0, 20.0, оператор: *
Результат: 200.0
200.0
```

Как работает код: Этот код реализует простой калькулятор с логированием и тестами.

1. Декоратор log_decorator: Оборачивает функцию calculate, логируя её имя, переданные аргументы и результат выполнения.
2. функция calculate: Выполняет арифметические операции (сложение, вычитание, умножение, деление) на основе переданных параметров.
3. Тестовые функции: Проверяют правильность работы calculate с различными входными данными.
4. Функция main: Запрашивает у пользователя ввод чисел и оператора, вызывает calculate и выводит результат.

Задание №3

Условие: Изучите основы каррирования. Каррирование в самом простом варианте - это создание специализированной функции на основе более общей функции с предустановленными параметрами для этой более общей функции. Реализуйте каррирование на примере вычисления количества радиоактивного вещества N, оставшегося в некоторый В качестве проставленного заранее параметра в данном примере должно быть значение периода полураспада t1/2, которое постоянно для каждого типа радиоактивного материала (радиоактивного изотопа химического элемента). Сделайте словарь, где в качестве колючей используются строки с символами радиоактивных изотопов, а в качестве значений им сопоставлены каррированные с характерными периодами полураспада. В основном коде вашей программы организуйте цикл по этому словарю и продемонстрируйте в нём вызовы каррированных функций с распечаткой на экране сколько вещества осталось от одного и того же N0 в некоторый момент времени t в зависимости от типа изотопа.

Код программы и результат:

```
def decay(N0, t, half_life):
    return N0 * (0.5 ** (t / half_life))

def curry(half_life):
    return lambda N0, t: decay(N0, t, half_life)

isotopes = {
    "C-14": 5730,
    "U-238": 4500000000,
    "K-40": 1261000000,
}

carr_funcs = {iso: curry(hl) for iso, hl in isotopes.items()}

N0 = 1000
t = 1000

for iso, func in carr_funcs.items():
    remaining = func(N0, t)
    print(f"{iso}: Остаток после {t} лет = {remaining:.2f}")
```

```
C-14: Остаток после 1000 лет = 886.06
U-238: Остаток после 1000 лет = 1000.00
K-40: Остаток после 1000 лет = 1000.00
```

Задание № 4

Условие : Напишите unit-тесты для калькулятора из задачи 3.1 лабораторной работы № 1 используя стандартный модуль unittest библиотеки Python. Базовый пример: KttpV docV.p\tKon.org liErar\ unitteVt. KtPl EaVic e[aPple Затем перепешите теже тесты с использованием пакета pytest. Ссылка на сайт библиотеки с базовым примером: KttpV p\teVt.org en .2.[.

Код программы и результат:

```
import logging
import functools

logger = logging.getLogger(__name__)

def log_call(func):
    @functools.wraps(func)
    def wrapper(*args, **kwargs):
        logger.info(f"Вызывается функция {func.__name__} с
параметрами: {args}, {kwargs}")
        result = func(*args, **kwargs)
        logger.info(f"Функция {func.__name__} вернула значение:
{result}")
        return result
    return wrapper

@log_call
```

```python
def calculate(num1, num2, operation):
    if operation == '+':
        return num1 + num2
    elif operation == '-':
        return num1 - num2
    elif operation == '/':
        if num2 != 0:
            return num1 / num2
        else:
            return "Делить на 0 нельзя!"
    elif operation == '*':
        return num1 * num2
    else:
        return "Недопустимая операция!"

def main():
    num1 = float(input("Введите первое число: "))
    num2 = float(input("Введите второе число: "))
    operation = input("Введите тип арифметической операции: ")
    result = calculate(num1, num2, operation)
    print(f"Результат: {result}")

def test_add():
    assert calculate(5, 5, "+") == 10

def test_subtract():
    assert calculate(5, 5, "-") == 0

def test_divide():
    assert calculate(5, 5, "/") == 1

def test_divide_by_zero():
    assert calculate(5, 0, "/") == "Делить на 0 нельзя!"

def test_multiply():
    assert calculate(5, 5, "*") == 25

def test_invalid_operation():
    assert calculate(5, 5, "%") == "Недопустимая операция!"

if __name__ == "__main__":
    main()
    test_add()
    test_subtract()
    test_divide()
    test_divide_by_zero()
    test_multiply()
    test_invalid_operation()
    print("Все тесты прошли успешно!")
```

```
Введите первое число: 10
Введите второе число: 20
Введите тип арифметической операции: *
Результат: 200.0
Все тесты прошли успешно!
```

```python
import functools
import unittest

def log_call(func):
    @functools.wraps(func)
    def wrapper(*args, **kwargs):
        print(f"Вызывается функция {func.__name__} с
параметрами: {args}, {kwargs}")
        result = func(*args, **kwargs)
        print(f"Функция {func.__name__} вернула значение:
{result}")
        return result
    return wrapper

@log_call
def calculate(num1, num2, operation):
    if operation == '+':
        return num1 + num2
    elif operation == '-':
        return num1 - num2
    elif operation == '/':
        if num2 != 0:
            return num1 / num2
        else:
            return "Делить на 0 нельзя!"
    elif operation == '*':
        return num1 * num2
    else:
        return "Недопустимая операция!"

def main():
    num1 = float(input("Введите первое число: "))
    num2 = float(input("Введите второе число: "))
    operation = input("Введите тип арифметической операции: ")
    result = calculate(num1, num2, operation)
    print(f"Результат: {result}")

class TestCalculator(unittest.TestCase):

    def test_add(self):
        self.assertEqual(calculate(5, 5, "+"), 10)

    def test_subtract(self):
```

```
        self.assertEqual(calculate(5, 5, "-"), 0)

    def test_divide(self):
        self.assertEqual(calculate(5, 5, "/"), 1)

    def test_divide_by_zero(self):
        self.assertEqual(calculate(5, 0, "/"), "Делить на 0
нельзя!")

    def test_multiply(self):
        self.assertEqual(calculate(5, 5, "*"), 25)

    def test_invalid_operation(self):
        self.assertEqual(calculate(5, 5, "%"), "Недопустимая
операция!")

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

```
Вызывается функция calculate с параметрами: (5, 5, '+'), {}
Функция calculate вернула значение: 10
Вызывается функция calculate с параметрами: (5, 5, '/'), {}
Функция calculate вернула значение: 1.0
Вызывается функция calculate с параметрами: (5, 0, '/'), {}
Функция calculate вернула значение: Делить на 0 нельзя!
Вызывается функция calculate с параметрами: (5, 5, '%'), {}
Функция calculate вернула значение: Недопустимая операция!
Вызывается функция calculate с параметрами: (5, 5, '*'), {}


Функция calculate вернула значение: 25
Ran 6 tests in 0.009s

OK
Вызывается функция calculate с параметрами: (5, 5, '-'), {}
Функция calculate вернула значение: 0
```

Как работает код:

1. Импорт библиотек: Использует functools для создания декоратора и unittest для юнит-тестов.

2. Декоратор log_call: Логирует вызовы функции и их результаты.

3. Функция calculate: Выполняет арифметические операции на двух числах.

4. main: Запрашивает у пользователя числа и операцию, затем выводит результат.

5. Класс TestCalculator: Содержит тесты для проверки правильности работы calculate.

6. Запуск: При запуске сначала выполняется main(), затем тесты.