# Лабораторная работа №2:

## Комплект 1: Начало использования Closures, Decorators, Logging, Unittests.

1.1: 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:

1.1.py
```python
def function_1(a, b):
    def function_2():
        print(a, b)

    #Возвращаем внутреннюю функцию
    return function_2

#Использование
function = function_1('Python', 'IITTO')
function()  # Выведет "Python IITTO"
```

### Результат:

```
PS C:\Users\sar
e/Рабочий стол/
Python IITTO
```

### Описание кода:

Написана функция «function_1», которая принимает на вход два аргумента. Внутри этой функции написана вторая функция «function_2», которая выводит (принтует) аргументы, которые принимаются 1 функцией. Создана переменная «function», в которую записана 1 функция. Далее вызвана 1 функция, через переменную.

1.2: 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:

1.2.py > test_subtract

```python
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
def calculate(num1, num2, operation):
    """Функция вычисления"""
    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

def test_add():
    """Тест операции сложения"""
    result = calculate(5, 5, "+")
    assert result == 10

def test_subtract():
    """Тест операции вычитания"""
    result = calculate(5, 5, "-")
    assert result == 0

```

```python
def test_divide():
    """Тест операции деления"""
    result = calculate(5, 5, "/")
    assert result == 1

def test_multiply():
    """Тест операции умножения"""
    result = calculate(5, 5, "*")
    assert result == 25

def main():
    # Ввод первого числа
    num1 = float(input("Введите первое число: "))

    # Ввод второго числа
    num2 = float(input("Введите второе число: "))

    # Ввод типа операции
    operation = input("Введите тип арифметической операции: ")

    # Вызов функции расчета и вывод результата
    result = calculate(num1, num2, operation)
    print(f"Результат: {result}")

if __name__ == "__main__":
    main()
    test_add()
    test_subtract()
    test_divide()
    test_multiply()
```

Результат:

```
Введите первое число: 10
Введите второе число: 2
Введите тип арифметической операции: /
Результат: 5.0
PS C:\Users\sambu\OneDrive\Рабочий стол\
```

Описание кода:

Использовано простое логирование для модернизации калькулятора из прошлой лабораторной работы.

1.3: 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 момент времени t от радокактивного вещества с периодом полураспада $t_{1/2}$, если изначально это количество было равно $N_0$. Закон распада задан формулой:

$$N = N_0 \cdot \left(\frac{1}{2}\right)^{t/t_{1/2}}$$

В качестве проставленного заранее параметра в данном примере должно быть значение периода полураспада $t_{1/2}$, 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 $N_0$ в некоторый момент времени t в зависимости от типа изотопа.

Код программы:

1.3.py > ...

```python
def remainder(N0, t, t_half):
    """Вычисляем остаток радиоактивного вещества."""
    return N0 * (1 / 2) ** (t / t_half)

def curry(t_half):
    """Создаем каррированную функцию с фиксированным периодом полураспада."""
    return lambda N0, t: remainder(N0, t, t_half)

#Словарь изотопов и их периодов полураспада
isotopes = {
    "C-14": 5730,
    "U-238": 4500000000,
    "K-40": 1261000000,
}

#Создаем словарь каррированных функций
carr_funcs = {isotope: curry(t_half) for isotope, t_half in isotopes.items()}

#Исходные данные
N0 = 1000
t = 1000

#Цикл по каррированным функциям и вывод результата
for isotope, func in carr_funcs.items():
    remaining = func(N0, t)
    print(f"{isotope}: Остаток после {t} лет = {remaining:.2f}")
```

**Результат:**

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

**Описание кода:**

Данный код содержит две функции: remainder и curry. Функция remainder вычисляет количество радиоактивного вещества после времени t, при начальном количестве $N_0$ и периоде полураспада t_half. Функция curry создает каррированную версию функции remainder с фиксированным периодом полураспада.

Далее создается словарь изотопов с их соответствующими периодами полураспада. Затем используется генераторное выражение для создания словаря каррированных функций, где каждая функция имеет фиксированный период полураспада соответствующего изотопа.

После этого задаются исходные данные и запускается цикл по каррированным функциям, который выводит остаток каждого изотопа после заданного количества лет.

1.4: Напишите unit-тесты для калькулятора из задачи 3.1 лабораторной работы № 1 используя стандартный модуль unittest библиотеки Python. Затем перепешите теже тесты с использованием пакета pytest.

**Код программы:**

На следующей странице

Unit-тесты:

1.4(unittest).py > ...

```python
import unittest

def calculate(num1, num2, operation):
    """Функция вычисления"""
    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

class CalculatorTestCase(unittest.TestCase):
    def test_add(self):
        result = calculate(5, 5, "+")
        self.assertEqual(result, 10)

    def test_subtract(self):
        result = calculate(5, 5, "-")
        self.assertEqual(result, 0)

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

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

def main():
    # Ввод первого числа
    num1 = float(input("Введите первое число: "))

    # Ввод второго числа
    num2 = float(input("Введите второе число: "))

    # Ввод типа операции
    operation = input("Введите тип арифметической операции: ")
```

```python
    # Вызов функции расчета и вывод результата
    result = calculate(num1, num2, operation)
    print(f"Результат: {result}")

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

**Результат:**

```
Введите первое число: 5
Введите второе число: 5
Введите тип арифметической операции: +
Результат: 10.0
....
----------------------------------------
Ran 4 tests in 0.001s

OK
PS C:\Users\sambu\OneDrive\Рабочий стол\
```

Pytest:

1.4.pytest.py > ...

```python
def calculate(num1, num2, operation):
    """Функция вычисления"""
    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

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_multiply():
    assert calculate(5, 5, "*") == 25

def main():
    # Ввод первого числа
    num1 = float(input("Введите первое число: "))

    # Ввод второго числа
    num2 = float(input("Введите второе число: "))

    # Ввод типа операции
    operation = input("Введите тип арифметической операции: ")

    # Вызов функции расчета и вывод результата
    result = calculate(num1, num2, operation)
    print(f"Результат: {result}")
```

```
40
41  if __name__ == "__main__":
42      main()
43      test_add()
44      test_subtract()
45      test_divide()
46      test_multiply()
```

**Результат:**

```
Введите первое число: 5
Введите второе число: 5
Введите тип арифметической операции: +
Результат: 10.0
PS C:\Users\sambu\OneDrive\Рабочий стол
```

**Описание кодов:**

Добавлены unit-тесты для калькулятора.

Pytest уже были реализованы при выполнении задания 3.1 в лабораторной работы №1.