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:

```
def func():
  '''Обычная функция для создания замыкания.
  Аргументы:
  name - некое имя;
  birthday - некая дата рождения;
  place - место рождения.
  Функция возвращает внутреннюю функцию.
  '''
  name = 'Natalya'
  birthday = '31.03.1789'
  place = 'Russia'

  def func_second():
    '''Внутренняя функция для создания замыкания.
    Возвращает словарь с именем, датой рождения и местом рождения.
    '''
    return {'name': name, 'birthday': birthday, 'place': place}

  return func_second


print(func()())
```

Результат программы:

```
{'name': 'Natalya', 'birthday': '31.03.1789', 'place': 'Russia'}
```

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:

```
import logging

logging.basicConfig(level=logging.DEBUG)

logger = logging.getLogger(__name__)

def log_decorator(func):
    '''Декорирование функции и его переменных.
    В декорирующем замыкании происходит логирование до и после вызова
функции.
    '''
    def log(oper1, oper2, operator):
        logger.debug(f'Введенные операнды и операция: {oper1}, {oper2},
{operator}')
        res = func(oper1, oper2, operator)
        logging.info(f'Результат: {res}')
        return res
    return log

@log_decorator
def calculate(a, b, operator):
  ''' Выполняет определенную арифметическую операцию над двумя числами.
  Если заданный операнд равен предложенным, то выполняется соответствующая
операция и возвращается результат вычисления.
  Аргументы:
  a - первое числло.
  b - второе число.
  operator - оператор, который задается и используется для выполнения
операции.
  return - возвращает результат выполнения арифметической операции.
  '''
  if operator == '+':
    return a + b
  elif operator == '-':
    return a - b
  elif operator == '*':
    return a * b
  elif operator == '/':
    if b == 0:
      return 'На ноль делить нельзя!'
    return a / b
  else:
    return 'Недопустимый оператор'


def test_calc():
    '''Проводит тесты на определенных примерах'''
    assert calculate(1, 2, '+') == 3
    assert calculate(4, 65, '-') == -61
    assert calculate(5, 5, '*') == 25
    assert calculate(10, 2, '/') == 5
    print('Все тесты пройдены')
```

```
def main():
  '''Основная функция.
  Аргументы:
  number1 - первое числло.
  number2 - второе число.
  operator - оператор, который задается и используется для выполнения
операции.
  '''
  number1 = float(input('Введите первое число: '))
  number2 = float(input('Введите второе число: '))
  operator = str(input('Введите оператор (+, -, *, /): '))

  print(calculate(number1, number2, operator))

main()
test_calc()
```

Результат программы:

```
Введите первое число: 56
Введите второе число: 33
Введите оператор (+, -, *, /): *
DEBUG:__main__:Введенные операнды и операция: 56.0, 33.0, *
INFO:root:Результат: 1848.0
1848.0
DEBUG:__main__:Введенные операнды и операция: 1, 2, +
INFO:root:Результат: 3
DEBUG:__main__:Введенные операнды и операция: 4, 65, -
INFO:root:Результат: -61
DEBUG:__main__:Введенные операнды и операция: 5, 5, *
INFO:root:Результат: 25
DEBUG:__main__:Введенные операнды и операция: 10, 2, /
INFO:root:Результат: 5.0
Все тесты пройдены
```

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 1 момент времени t от радокактивного вещества с периодом полураспада t1/2, если изначально это количество было равно N0. Закон распада задан формулой:

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

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:

```
from functools import partial

t1_2_elems = {"U_234": 7.74 * 10**12, "U_235": 2.22 * 10**16, "U_238": 1.41 *
10**17}
radioactive_funcs = {"U_234": None, "U_235": None, "U_238": None}

def decay_amount(N0, t, t1_2):
    '''Вычисляет количество радиоактивного вещества, оставшегося в некоторый
момент t по закону распада.
    Аргументы:
    N0 - изначальное количества вещества;
    t - некоторый момент времени;
    t1_2 - период полураспада вещества.
    '''
    N = N0 * (1/2) ** (t/t1_2)
    res = "Масса радиоактивного вещества, t1_2=" + str(t1_2)
    print(f'{res} с периодом полураспада {t1_2}, N0 = {N0}, t = {t}')
    return N

f1 = partial(decay_amount, t1_2 = t1_2_elems['U_234'])
f2 = partial(decay_amount, t1_2 = t1_2_elems['U_235'])
f3 = partial(decay_amount, t1_2 = t1_2_elems['U_238'])

def main():
    '''Основной код программы, где вызываются каррированные функции.
    И организован цикл по словарю с распечаткой на экране сколько вещества
осталось от одного и того же N0 в некоторый момент времени t.
    '''
    N0 = 100
    t = 4006.9
    radioactive_funcs["U_234"] = f1
    radioactive_funcs["U_235"] = f2
    radioactive_funcs["U_238"] = f3

    for isotope, func in radioactive_funcs.items():
        result = func(N0, t)
        print(f'Изотоп: {isotope}, Остаток: {result}')


main()
```

Результат программы:

```
Масса радиоактивного вещества, t1_2=7740000000000.0 с периодом полураспада 7740000000000.0, N0 = 100, t = 4006.9
Изотоп: U_234, Остаток: 99.99999996411665
Масса радиоактивного вещества, t1_2=2.22e+16 с периодом полураспада 2.22e+16, N0 = 100, t = 4006.9
Изотоп: U_235, Остаток: 99.9999999999875
Масса радиоактивного вещества, t1_2=1.41e+17 с периодом полураспада 1.41e+17, N0 = 100, t = 4006.9
Изотоп: U_238, Остаток: 99.99999999999804
```

1.4: Напишите unit-тесты для калькулятора из задачи 3.1 лабораторной работы № 1 используя стандартный модуль unittest библиотеки Python. Базовый пример: https://docs.python.org/3/library/unittest.html#basic-example

Затем перепешите теже тесты с использованием пакета pytest. Ссылка на сайт библиотеки с базовым примером: https://pytest.org/en/7.2.x/

Код программы:

Задача 1 из Лабораторной работы № 1

```
def calculate(a: float, b: float, operator: str):
  ''' Выполняет определенную арифметическую операцию над двумя числами.
  Если заданный операнд равен предложенным, то выполняется соответствующая
операция и возвращается результат вычисления.
  Аргументы:
  a - первое числло.
  b - второе число.
  operator - оператор, который задается и используется для выполнения
операции.
  return - возвращает результат выполнения арифметической операции.
  '''
  if operator == '+':
    return a + b
  elif operator == '-':
    return a - b
  elif operator == '*':
    return a * b
  elif operator == '/':
    if b == 0:
      return 'На ноль делить нельзя!'
    return a / b
  else:
    return 'Недопустимый оператор'


def test_add():
  '''Проверяет функцию calculate() на сложения чисел через тестирование.'''
  assert calculate(1, 2, '+') == 3
  return 'test passed'


def test_sub():
  '''Проверяет функцию calculate() на вычитание чисел через тестирование.'''
  assert calculate(4, 65, '-') == -61
  return 'test passed'
```

```
def test_mult():
  '''Проверяет функцию calculate() на умножение чисел через тестирование.'''
  assert calculate(5, 5, '*') == 25
  return 'test passed'


def test_div():
  '''Проверяет функцию calculate() на деление чисел через тестирование.'''
  assert calculate(10, 2, '/') == 5
  return 'test passed'


def main():
  '''Основная функция.
  Аргументы:
  number1 - первое числло.
  number2 - второе число.
  operator - оператор, который задается и используется для выполнения
операции.
  '''
  # number1 = float(input('Введите первое число: '))
  # number2 = float(input('Введите второе число: '))
  # operator = str(input('Введите оператор: '))

  # print(calculate(number1, number2, operator))
  print(test_add())
  print(test_sub())
  print(test_div())
  print(test_mult())

main()
```

1.4.

```
import unittest
import LR2_code3
import LR1_3_1


class Testprogramm3(unittest.TestCase):
    '''Тесты для задания 3, в которой тестируются функции радиоактивного
распада, взятые из модуля LR2 code3.py.'''

    def test_uranium_234(self):
        self.assertEqual(LR2_code3.radioactive_funcs["U_234"](100, 3850.9),
99.99999996551368)

    def test_uranium_235(self):
        self.assertEqual(LR2_code3.radioactive_funcs["U_235"](100, 3980.9),
99.99999999998757)

    def test_uranium_238(self):
        self.assertEqual(LR2_code3.radioactive_funcs["U_238"](100, 4006.9),
99.99999999999804)


class Testprogrammlr11(unittest.TestCase):
    '''Тесты для задания 1 ЛР 1, в которой тестируются функции для
тестирования арефмитических действий, взятые из модуля LR1_3_1.py.'''

    def test_add(self):
```

```
        self.assertEqual(LR1_3_1.calculate(1, 2, '+'), 3)

    def test_sub(self):
        self.assertEqual(LR1_3_1.calculate(4, 65, '-'), -61)

    def test_mult(self):
        self.assertEqual(LR1_3_1.calculate(5, 5, '*'), 25)

    def test_div(self):
        self.assertEqual(LR1_3_1.calculate(10, 2, '/'), 5)

    def test_div_0(self):
        self.assertEqual(LR1_3_1.calculate(10, 0, '/'), 'На ноль делить
нельзя!')

if   name   == '  main  ':
    unittest.main()
```

Результат программы:

Unit-тест

```
Testing started at 15:33 ...
Launching pytest with arguments C:\Users\melni\Desktop\ВУ3\2 курс\Программирование ру\ЛР 2\LR2_code4.py --no-header --no-summ

============================= test session starts =============================
collecting ... collected 8 items

LR2_code4.py::Testprogramm3::test_uranium_234 PASSED                  [ 12%]Масса радиоактивного вещества, t1_2=7740000000

LR2_code4.py::Testprogramm3::test_uranium_235 PASSED                  [ 25%]Масса радиоактивного вещества, t1_2=2.22e+16 с

LR2_code4.py::Testprogramm3::test_uranium_238 PASSED                  [ 37%]Масса радиоактивного вещества, t1_2=1.41e+17 с

LR2_code4.py::Testprogrammlr11::test_add PASSED                       [ 50%]
LR2_code4.py::Testprogrammlr11::test_div PASSED                       [ 62%]
LR2_code4.py::Testprogrammlr11::test_div_0 PASSED                     [ 75%]
LR2_code4.py::Testprogrammlr11::test_mult PASSED                      [ 87%]
LR2_code4.py::Testprogrammlr11::test_sub PASSED                       [100%]

============================== 8 passed in 0.08s ==============================

Process finished with exit code 0
```

pytest

```
============================= test session starts =============================
platform win32 -- Python 3.12.7, pytest-8.3.3, pluggy-1.5.0
rootdir: C:\Users\melni\Desktop\ВУ3\2 курс\Программирование ру\ЛР 2
collected 8 items

LR2_code4.py ........                                                    [100%]

============================== 8 passed in 0.04s ==============================
```