Программирование. Язык Python.

Лабораторная работа № 2. Задачи. 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 foo():
  name = 'Ann'

  surname = 'Stepanova'

  patronymic = 'Andreevna'

  birthday = '20.04.2005'

  def inner_boo():

    return {
        'name': name,
        'surname': surname,
        'patronymic': patronymic,
        'birthday': birthday
    }

  return inner_boo


print(foo()())
```

Результат:

```
{'name': 'Ann', 'surname': 'Stepanova', 'patronymic': 'Andreevna', 'birthday': '20.04.2005'}
```

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

# Настройка логирования
```

```python
logging.basicConfig(level=logging.INFO,
                    format='%(asctime)s - %(levelname)s - %(message)s')


def log_decorator(func):

  def wrapper(a, b, operand):
    # Логирование входных данных
    logging.info(f"Посчитать: {a} {operand} {b}")

    # Вызов оригинальной функции
    result = func(a, b, operand)

    # Логирование результата
    logging.info(f"Результат: {result}")

    return result

  return wrapper


@log_decorator
def calculate(a, b, operand):
  """
  Функция вычисления

  Параметры:
  a (int): Первое число
  b (int): Второе число
  operand (str): Операция над числами

  Возвращает:
  int: Результат вычиислений
  или str: сообщение об ошибке при делении на ноль

  """

  if operand == '+':
    return a + b
  if operand == '-':
    return a - b
  if operand == '*':
    return a * b
  elif operand == '/':
    if b == 0:
      return 'Ошибка: на ноль делить нельзя'
    else:
      return a / b


def test_calculator():
  """
  Функция для тестирования функции calculate

  Проверяет корректность выполнения операций сложения, вычитания,
  умножения и деления, а также обработку ошибок

  """
  assert calculate(5, 3, '+') == 8
  assert calculate(8, 3, '-') == 5
  assert calculate(7, 6, '*') == 42
  assert calculate(3, 3, '/') == 1
```

```
test_calculator()
```

Результат:

```
2024-10-17 15:38:24,417 - INFO - Посчитать: 5 + 3
2024-10-17 15:38:24,417 - INFO - Результат: 8
2024-10-17 15:38:24,417 - INFO - Посчитать: 8 - 3
2024-10-17 15:38:24,417 - INFO - Результат: 5
2024-10-17 15:38:24,417 - INFO - Посчитать: 7 * 6
2024-10-17 15:38:24,417 - INFO - Результат: 42
2024-10-17 15:38:24,417 - INFO - Посчитать: 3 / 3
2024-10-17 15:38:24,417 - INFO - Результат: 1.0
```

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:

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

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

# Периоды полураспада в секундах
half_per = {
    "Po_210": 138.4 * 24 * 3600,
    "Po_214": 0.16,
}

radioactive_funcs = {"Po_210": None, "Po_214": None}
```

```python
def decay_amount(N0: float, t: int, t1_2 = None):

    """
    Функция закона полураспада

    Параметры:
    N0 (float): изначальное количества вещества
    t (int): некоторый момент времени
    t1_2 (NoneType): период полураспада вещества

    """

    N = N0 * (1 / 2) ** (t / t1_2)

    #res = 'Масса радиоактивного вещества:' + str(t1_2)
    print(f'Масса радиоактивного вещества: {str(t1_2)} \nПериод полураспада
{t1_2}, \nN0 = {N0}, t = {t} c')
    return N


f1 = partial(decay_amount, t1_2 = half_per['Po_210'])
f2 = partial(decay_amount, t1_2 = half_per['Po_214'])


def main():

    """
    Вызов каррирования функции
    Создан цикл по словарю с распечаткой на экране сколько вещества осталось
от одного и того же N0 в момент времени t
    """

    N0 = 100
    t = 150


    radioactive_funcs["Po_210"] = f1
    radioactive_funcs["Po_214"] = f2

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


main()
```

Результат:

```
Масса радиоактивного вещества: 11957760.000000002
Период полураспада 11957760.000000002,
N0 = 100, t = 150 c
Изотоп: Po_210, Остаток: 99.9991305091834
Масса радиоактивного вещества: 0.16
Период полураспада 0.16,
N0 = 100, t = 150 c
Изотоп: Po_214, Остаток: 6.08666031138343e-281
```

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:

```
import unittest
import Calculate
import T3


class TestStringMethods(unittest.TestCase):


    def test_polonium_210(self):
        self.assertEqual(T3.radioactive_funcs["Po_210"](100, 150),
99.9991305091834)

    def test_polonium_214(self):
        self.assertEqual(T3.radioactive_funcs["Po_214"](100, 150),
6.08666031138343e-281)



    def test_summ(self):
        self.assertEqual(Calculate.calculate(5, 3, '+'), 8)

    def test_div(self):
        self.assertEqual(Calculate.calculate(3, 3, '/'), 1)

    def test_mult(self):
        self.assertEqual(Calculate.calculate(3, 9, '*'), 0)



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

Результат:

В тесте калькулятора специально была сделана ошибка

```
============================= test session starts ==============================
collecting ... collected 5 items

T4.py::TestStringMethods::test_div PASSED                                [ 20%]
T4.py::TestStringMethods::test_mult FAILED                               [ 40%]
T4.py:22 (TestStringMethods.test_mult)
0 != 27

Expected :27
Actual   :0
<Click to see difference>

self = <T4.TestStringMethods testMethod=test_mult>

    def test_mult(self):
```

```
    def test_mult(self):
>       self.assertEqual(Calculate.calculate(3, 9, '*'), 0)

T4.py:24: AssertionError

T4.py::TestStringMethods::test_polonium_210 PASSED                       [ 60%]Масса радиоактивного вещества: 11957760.000000002
Период полураспада 11957760.000000002,
N0 = 100, t = 150 c

T4.py::TestStringMethods::test_polonium_214 PASSED                       [ 80%]Масса радиоактивного вещества: 0.16
Период полураспада 0.16,
N0 = 100, t = 150 c

T4.py::TestStringMethods::test_summ PASSED                               [100%]

========================= 1 failed, 4 passed in 0.06s =========================
```