

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 > ...
1  def function_1(a, b):
2      def function_2():
3          print(a, b)
4
5      #Возвращаем внутреннюю функцию
6      return function_2
7
8  #Использование
9  function = function_1('Python', 'IITTO')
10 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 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, сделанного в этой функции.

Код программы:

```
1.2.py > test_subtract
1  import logging
2  import functools
3
4  logger = logging.getLogger(__name__)
5
6  def log_call(func):
7      @functools.wraps(func)
8      def wrapper(*args, **kwargs):
9          logger.info(f"Вызывается функция {func.__name__} с параметрами: {args}, {kwargs}")
10         result = func(*args, **kwargs)
11         logger.info(f"Функция {func.__name__} вернула значение: {result}")
12         return result
13     return wrapper
14
15 @log_call
16 def calculate(num1, num2, operation):
17     """Функция вычисления"""
18     if operation == '+':
19         return num1 + num2
20     elif operation == '-':
21         return num1 - num2
22     elif operation == '/':
23         if num2 != 0:
24             return num1 / num2
25         else:
26             return "Делить на 0 нельзя!"
27     elif operation == '*':
28         return num1 * num2
29
30 def test_add():
31     """Тест операции сложения"""
32     result = calculate(5, 5, "+")
33     assert result == 10
34
35 def test_subtract():
36     """Тест операции вычитания"""
37     result = calculate(5, 5, "-")
38     assert result == 0
39
```

```

40 def test_divide():
41     """Тест операции деления"""
42     result = calculate(5, 5, "/")
43     assert result == 1
44
45 def test_multiply():
46     """Тест операции умножения"""
47     result = calculate(5, 5, "*")
48     assert result == 25
49
50 def main():
51     # Ввод первого числа
52     num1 = float(input("Введите первое число: "))
53
54     # Ввод второго числа
55     num2 = float(input("Введите второе число: "))
56
57     # Ввод типа операции
58     operation = input("Введите тип арифметической операции: ")
59
60     # Вызов функции расчета и вывод результата
61     result = calculate(num1, num2, operation)
62     print(f"Результат: {result}")
63
64 if __name__ == "__main__":
65     main()
66     test_add()
67     test_subtract()
68     test_divide()
69     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 радиоактивного 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 ключей 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 словарию 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 > ...
1 def remainder(N0, t, t_half):
2     """Вычисляем остаток радиоактивного вещества."""
3     return N0 * (1 / 2) ** (t / t_half)
4
5 def curry(t_half):
6     """Создаем каррированную функцию с фиксированным периодом полураспада."""
7     return lambda N0, t: remainder(N0, t, t_half)
8
9 #Словарь изотопов и их периодов полураспада
10 isotopes = {
11     "C-14": 5730,
12     "U-238": 4500000000,
13     "K-40": 1261000000,
14 }
15
16 #Создаем словарь каррированных функций
17 carr_funcs = {isotope: curry(t_half) for isotope, t_half in isotopes.items()}
18
19 #Исходные данные
20 N0 = 1000
21 t = 1000
22
23 #Цикл по каррированным функциям и вывод результата
24 for isotope, func in carr_funcs.items():
25     remaining = func(N0, t)
26     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 перепишите те же тесты с использованием пакета `pytest`.

Код программы:

На следующей странице



Unit-тесты:

```
1.4(unittest).py > ...
1 import unittest
2 import logging
3 import functools
4
5 logger = logging.getLogger(__name__)
6
7 def log_call(func):
8     @functools.wraps(func)
9     def wrapper(*args, **kwargs):
10         logger.info(f"Вызывается функция {func.__name__} с параметрами: {args}, {kwargs}")
11         result = func(*args, **kwargs)
12         logger.info(f"Функция {func.__name__} вернула значение: {result}")
13         return result
14     return wrapper
15
16 @log_call
17 def calculate(num1, num2, operation):
18     """Функция вычисления"""
19     if operation == '+':
20         return num1 + num2
21     elif operation == '-':
22         return num1 - num2
23     elif operation == '/':
24         if num2 != 0:
25             return num1 / num2
26         else:
27             return "Делить на 0 нельзя!"
28     elif operation == '*':
29         return num1 * num2
30
31 class CalculatorTestCase(unittest.TestCase):
32     def test_add(self):
33         result = calculate(5, 5, "+")
34         self.assertEqual(result, 10)
35
36     def test_subtract(self):
37         result = calculate(5, 5, "-")
38         self.assertEqual(result, 0)
39
40     def test_divide(self):
41         result = calculate(5, 5, "/")
42         self.assertEqual(result, 1)
```

```
43
44     def test_multiply(self):
45         result = calculate(5, 5, "*")
46         self.assertEqual(result, 25)
47
48 def main():
49     # Ввод первого числа
50     num1 = float(input("Введите первое число: "))
51
52     # Ввод второго числа
53     num2 = float(input("Введите второе число: "))
54
55     # Ввод типа операции
56     operation = input("Введите тип арифметической операции: ")
57
58     # Вызов функции расчета и вывод результата
59     result = calculate(num1, num2, operation)
60     print(f"Результат: {result}")
61
62 if __name__ == "__main__":
63     main()
64     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\Учеба в Герцена Алдар\Программи
```

Pytest:

```
1.4.pytest.py > log_call
1  import logging
2  import functools
3
4  logger = logging.getLogger(__name__)
5
6  def log_call(func):
7      @functools.wraps(func)
8      def wrapper(*args, **kwargs):
9          logger.info(f"Вызывается функция {func.__name__} с параметрами: {args}, {kwargs}")
10         result = func(*args, **kwargs)
11         logger.info(f"Функция {func.__name__} вернула значение: {result}")
12         return result
13     return wrapper
14
15 @log_call
16 def calculate(num1, num2, operation):
17     """Функция вычисления"""
18     if operation == '+':
19         return num1 + num2
20     elif operation == '-':
21         return num1 - num2
22     elif operation == '/':
23         if num2 != 0:
24             return num1 / num2
25         else:
26             return "Делить на 0 нельзя!"
27     elif operation == '*':
28         return num1 * num2
29
30 def test_add():
31     assert calculate(5, 5, "+") == 10
32
33 def test_subtract():
34     assert calculate(5, 5, "-") == 0
35
36 def test_divide():
37     assert calculate(5, 5, "/") == 1
38
39 def test_multiply():
40     assert calculate(5, 5, "*") == 25
```

```

42 def main():
43     # Ввод первого числа
44     num1 = float(input("Введите первое число: "))
45
46     # Ввод второго числа
47     num2 = float(input("Введите второе число: "))
48
49     # Ввод типа операции
50     operation = input("Введите тип арифметической операции: ")
51
52     # Вызов функции расчета и вывод результата
53     result = calculate(num1, num2, operation)
54     print(f"Результат: {result}")
55
56 if __name__ == "__main__":
57     main()
58     test_add()
59     test_subtract()
60     test_divide()
61     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.