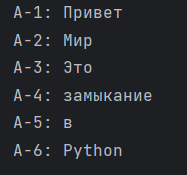
Лабораторная работа 1

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"А-{index + 1}: {value}")  
  
 return func  
  
c = main("Привет", "Мир", "Это", "замыкание", "в", "Python")  
c()



Как работает код:

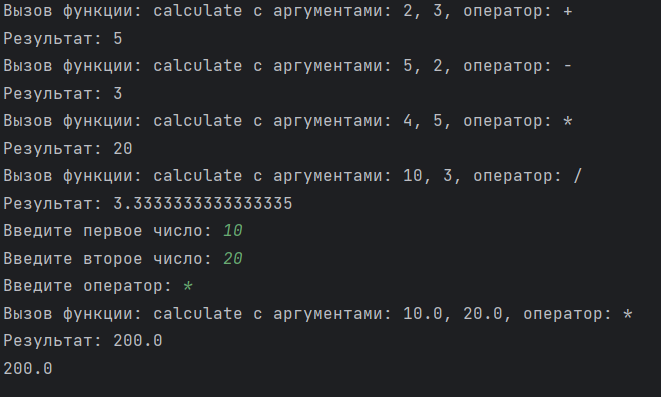
Код определяет функцию main, которая принимает произвольные аргументы. Внутри неё создаётся функция func, выводящая каждый аргумент с его индексом. main возвращает func, а вызов c() выводит строки с индексами.

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 сlosure, 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()

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

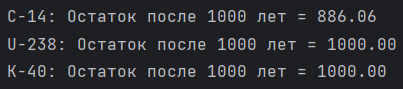
1. Декоратор log\_decorator: Оборачивает функцию calculate, логируя её имя, переданные аргументы и результат выполнения.
2. функция calculate: Выполняет арифметические операции (сложение, вычитание, умножение, деление) на основе переданных параметров.
3. Тестовые функции: Проверяют правильность работы 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}")

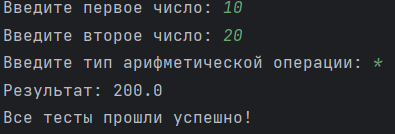


Задание № 4

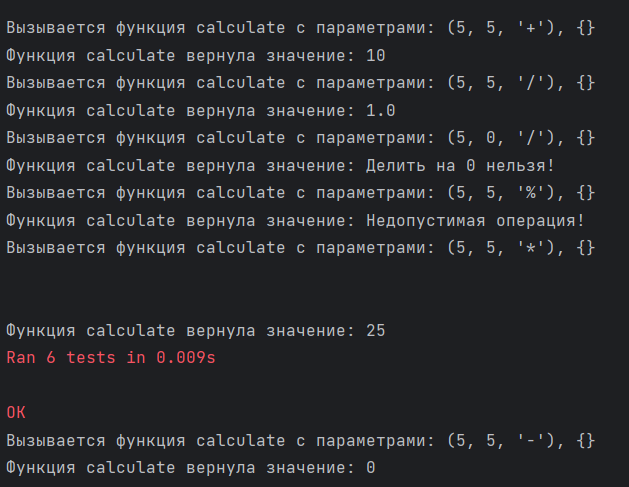
Условие : Напишите unit-тесты для калькулятора из задачи 3.1 лабораторной работы № 1 используя стандартный модуль unittest библиотеки Python. Базовый пример: KttpVdocV.p\tKon.orgliErar\unitteVt. KtPlEaVice[aPple Затем перепешите теже тесты с использованием пакета pytest. Ссылка на сайт библиотеки с базовым примером: KttpVp\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  
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!")



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()



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.