Лабораторная работа №3:

Комплект 1: Работа над мини-проектом калькулятор. Функции. Тесты

1.1: Модернизируйте калькулятор из задач 1.2 и 1.4 Лабораторной работы № 2. Добавьте к калькулятору такую настройку как точность вычислений, которая передаётся в виде keyword параметра tolerance со значением по умолчанию 1e−6. На основе переданного значения этого параметра извлеките с помощью вычислений порядок этого значения (например 6 для 1e−6) в виде отдельной функции convert_precision, вызываемой из calculate. Задокументируйте convert_precision и дополните документацию к calculate в коде. Извлечённый порядок используйте для округления итогового результата в функции calculate. Покройте (напишите) дополнительными тестами convert_precision и calculate в связи с появлением tolerance с помощью пакета pytest или стандартных unittest Python по выбору.

Код программы:

1.1.py > calculate

```python
import math
import unittest
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, tolerance = 1e-6):
    """Функция вычисления с точностью до определенного порядка."""
    order = convert_precision(tolerance)
    if operation == '+':
        return round(num1 + num2, order)
    elif operation == '-':
        return round(num1 - num2, order)
    elif operation == '/':
        if abs(num2) > tolerance:
            return round(num1 / num2, order)
        else:
            return "Делить на 0 нельзя!"
    elif operation == '*':
        return round(num1 * num2, order)

def convert_precision(tolerance):
    """Извлекает порядок точности из числа."""
    return int(-math.floor(math.log10(abs(tolerance))))

class CalculatorTestCase(unittest.TestCase):
    def test_add(self):
        result = calculate(5.5, 5.6, "+", tolerance=1e-1)
        self.assertAlmostEqual(result, 11.1, places=8)
```

```
    def test_subtract(self):
        result = calculate(5, 5, "-", tolerance=1e-4)
        self.assertAlmostEqual(result, 0, places=4)

    def test_divide(self):
        result = calculate(5, 5, "/", tolerance=1e-7)
        self.assertAlmostEqual(result, 1, places=7)

    def test_multiply(self):
        result = calculate(5, 5, "*", tolerance=1e-9)
        self.assertAlmostEqual(result, 25, places=9)

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: ")

    # Ввод точности вычислений
    tolerance = float(input("Введите точность вычислений (в формате 0.1): "))

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

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

Результат:

```
Введите первое число: 5
Введите второе число: 8
Введите тип арифметической операции: +
Введите точность вычислений (в формате 0.1): 0.1
Результат: 13.0
....
----------------------------------------------------------------------
Ran 4 tests in 0.001s

OK
```

Описание кода:

Добавлен ввод точности вычислений.

1.2: Модернизируйте калькулятор из задачи 1.1. Добавьте переменное количество неименованных аргументов (операндов, *args) после параметра action и перед keyword параметром tolerance. К списку поддерживаемых действий добавьте вычисление таких величин как среднее значение (medium), дисперсия (variance), стандартное отклонение (std_deviation), медиана (median, q2, второй квартиль) и межквартильный размах (q3 - q1, разница третьего и первого квартилей). Покройте новые реализованные функции и функцию calculate дополнительными юнит-тестами.

Код программы:

1.2.py > main

```python
import math
import unittest
import logging
from statistics import mean, variance, stdev, median
from functools import wraps


logging.basicConfig(level=logging.INFO,
                    format='%(asctime)s - %(levelname)s - %(message)s')
logger = logging.getLogger(__name__)

def log_call(func):
    @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(action, *args, tolerance=1e-6):
    """Функция вычисления с точностью до определенного порядка."""
    order = convert_precision(tolerance)
    if action == '+':
        return round(sum(args), order)
    elif action == '-':
        return round(args[0] - sum(args[1:]), order)
    elif action == '/':
        try:
            return round(args[0] / args[1], order)
        except ZeroDivisionError:
            return "Делить на 0 нельзя!"
    elif action == '*':
        product = 1
        for arg in args:
            product *= arg
        return round(product, order)
    elif action == 'mean':
        return round(mean(args), order)
    elif action == 'variance':
        return round(variance(args), order)
```

```
    elif action == 'std_deviation':
        return round(stdev(args), order)
    elif action == 'median':
        return round(median(args), order)
    elif action == 'iqr':  # Межквартильный размах
        sorted_args = sorted(args)
        q1 = median(sorted_args[:len(sorted_args)//2])
        q3 = median(sorted_args[len(sorted_args+1)//2:])
        return round(q3 - q1, order)
    else:
        raise ValueError(f'Неизвестная операция "{action}"')

def convert_precision(tolerance):
    """Извлекает порядок точности из числа."""
    return int(-math.floor(math.log10(abs(tolerance))))

class CalculatorTestCase(unittest.TestCase):
    def test_add(self):
        result = calculate('+', 5.5, 5.6, tolerance=1e-1)
        self.assertAlmostEqual(result, 11.1, places=8)

    def test_subtract(self):
        result = calculate('-', 5, 5, tolerance=1e-4)
        self.assertAlmostEqual(result, 0, places=4)

    def test_divide(self):
        result = calculate('/', 5, 5, tolerance=1e-7)
        self.assertAlmostEqual(result, 1, places=7)

    def test_multiply(self):
        result = calculate('*', 5, 5, tolerance=1e-9)
        self.assertAlmostEqual(result, 25, places=9)

    def test_mean(self):
        result = calculate('mean', 1, 2, 3, 4, 5, tolerance=1e-2)
        self.assertAlmostEqual(result, 3, places=2)
```

(variable) sorted_args: list

```
80          def test_variance(self):
81              result = calculate('variance', 1, 2, 3, 4, 5, tolerance=1e-2)
82              self.assertAlmostEqual(result, 2.5, places=2)
83
84          def test_std_deviation(self):
85              result = calculate('std_deviation', 1, 2, 3, 4, 5, tolerance=1e-2)
86              self.assertAlmostEqual(result, 1.58, places=2)
87
88          def test_median(self):
89              result = calculate('median', 1, 2, 3, 4, 5, tolerance=1e-2)
90              self.assertAlmostEqual(result, 3, places=2)
91
92          def test_iqr(self):
93              result = calculate('iqr', 1, 2, 3, 4, 5, tolerance=1e-2)
94              self.assertAlmostEqual(result, 2, places=2)
95
96      def main():
97          while True:
98              try:
99                  num1 = float(input("Введите первое число: "))
100                 break
101             except ValueError:
102                 print("Ошибка! Введено неверное значение.")
103
104         while True:
105             try:
106                 num2 = float(input("Введите второе число: "))
107                 break
108             except ValueError:
109                 print("Ошибка! Введено неверное значение.")
110
```

```
111     while True:
112         action = input("Введите тип арифметической операции (+, -, *, /, mean, variance, std_deviation, median, iqr): ").strip().lower()
113         if action in ['+', '-', '*', '/', 'mean', 'variance', 'std_deviation', 'median', 'iqr']:
114             break
115         else:
116             print("Ошибка! Неверный тип операции.")
117
118     while True:
119         try:
120             tolerance = float(input("Введите точность вычислений (в формате 0.1): "))
121             break
122         except ValueError:
123             print("Ошибка! Введено неверное значение.")
124
125     result = calculate(action, num1, num2, tolerance=tolerance)
126     print(f"Результат: {result}")
127
128 if __name__ == "__main__":
129     main()
130     unittest.main()
```

Результат:

```
Введите первое число: 8.54
Введите второе число: 4.65
Введите тип арифметической операции (+, -, *, /, mean, variance, std_deviation, median, iqr): variance
Введите точность вычислений (в формате 0.1): 0.1
2024-12-24 19:40:33,243 - INFO - Вызывается функция calculate с параметрами: ('variance', 8.54, 4.65), {'tolerance': 0.1}
2024-12-24 19:40:33,244 - INFO - Функция calculate вернула значение: 7.6
Результат: 7.6
2024-12-24 19:40:33,262 - INFO - Вызывается функция calculate с параметрами: ('+', 5.5, 5.6), {'tolerance': 0.1}
2024-12-24 19:40:33,262 - INFO - Функция calculate вернула значение: 11.1
.2024-12-24 19:40:33,263 - INFO - Вызывается функция calculate с параметрами: ('/', 5, 5), {'tolerance': 1e-07}
2024-12-24 19:40:33,264 - INFO - Функция calculate вернула значение: 1.0
.2024-12-24 19:40:33,265 - INFO - Вызывается функция calculate с параметрами: ('iqr', 1, 2, 3, 4, 5), {'tolerance': 0.01}
E2024-12-24 19:40:33,267 - INFO - Вызывается функция calculate с параметрами: ('mean', 1, 2, 3, 4, 5), {'tolerance': 0.01}
2024-12-24 19:40:33,267 - INFO - Функция calculate вернула значение: 3
.2024-12-24 19:40:33,268 - INFO - Вызывается функция calculate с параметрами: ('median', 1, 2, 3, 4, 5), {'tolerance': 0.01}
2024-12-24 19:40:33,268 - INFO - Функция calculate вернула значение: 3
.2024-12-24 19:40:33,269 - INFO - Вызывается функция calculate с параметрами: ('*', 5, 5), {'tolerance': 1e-09}
2024-12-24 19:40:33,270 - INFO - Функция calculate вернула значение: 25
.2024-12-24 19:40:33,270 - INFO - Вызывается функция calculate с параметрами: ('std_deviation', 1, 2, 3, 4, 5), {'tolerance': 0.01}
2024-12-24 19:40:33,271 - INFO - Функция calculate вернула значение: 1.58
.2024-12-24 19:40:33,271 - INFO - Вызывается функция calculate с параметрами: ('-', 5, 5), {'tolerance': 0.0001}
2024-12-24 19:40:33,272 - INFO - Функция calculate вернула значение: 0
.2024-12-24 19:40:33,273 - INFO - Вызывается функция calculate с параметрами: ('variance', 1, 2, 3, 4, 5), {'tolerance': 0.01}
2024-12-24 19:40:33,273 - INFO - Функция calculate вернула значение: 2.5
.
======================================================================
ERROR: test_iqr (__main__.CalculatorTestCase.test_iqr)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "c:\Users\sambu\OneDrive\Рабочий стол\Учеба в Герцена Алдар\II курс\Программирование Python, 2 курс\ЛР-3\1.2.py", line 93, in test_iqr
    result = calculate('iqr', 1, 2, 3, 4, 5, tolerance=1e-2)
  File "c:\Users\sambu\OneDrive\Рабочий стол\Учеба в Герцена Алдар\II курс\Программирование Python, 2 курс\ЛР-3\1.2.py", line 16, in wrapper
    result = func(*args, **kwargs)
  File "c:\Users\sambu\OneDrive\Рабочий стол\Учеба в Герцена Алдар\II курс\Программирование Python, 2 курс\ЛР-3\1.2.py", line 50, in calculate
    q3 = median(sorted_args[len(sorted_args+1)//2:])
                            ~~~~~~~~~~~~~~~~~~~~~~~
TypeError: can only concatenate list (not "int") to list

----------------------------------------------------------------------
Ran 9 tests in 0.012s
```

Описание кода:

Добавлено вычисление таких величин как среднее значение, дисперсия, стандартное отклонение, медиана и межквартильный размах. Добавлены юнит-тесты по новым функциям.