Лабораторная работа 1. Калькулятор с дополнительными действиями

Оставьте ссылку на выполненное задание ЛР 1 (ссылка на replit), где необходимо создать еще несколько действий для вычисления (2-3 действия) с двумя операндами.

Лабораторная работа 2. Калькулятор с настройками. Ср. квадратическое отклонение

Оставьте ссылку на выполненное задание ЛР 2 (ссылка на replit). Описание - см. борд: https://moodle.herzen.spb.ru/mod/url/view.php?id=825124

Лабораторная работа 3. Задача two_sum, two_sum_hashed

Напишите в поле ответа ссылку на собственное решение в repl.it.

Не забудьте указать в коде решения автора и написать тесты

Лабораторная работа 4. Тестирование

Цель работы

Освоить основные принципы модульного тестирования и базовый инструментарий обработки исключений.

Запись конференции

Код доступа: J6^WHje?

Комментарии по выполнению

Работу можно структурировать на следующие части:

1. Проанализировать ситуации, в которых может возникнуть исключение и реализовать обработку этих исключительных ситуаций с помощью базового инструментария, показанного в конспекте курса или по ссылкам (официальная документация, русско-язычный ресурс и ещё один по обработке исключений).
2. Создать набор тестов для с использованием оператора assert для тестирования функций **two_sum**, **convert_precision**, функции для вычисления **среднеквадратического отклонения, функции calculate**. Для этого:
   0) проанализировать функции и их ОДЗ, выявить краевые случаи для тестов, выявить какие-либо еще ситуации, которые не связаны с ОДЗ (например, с передачей значений некорректного типа данных);
   1) создать в repl.it бордах отдельные файлы, начинающиеся со слова "test_" и содержащие в названии имя тестируемой функции;
   2) создать внутри функции (также начинающиеся со слова test_и содержащие в

названии описание тестового случая) и использовать в них - assert, написать проверку ожидаемого результата.

3. Применить принципы модульного тестирования и с использованием библиотеки unittest (см. пример в repl.it, сайт с официальной документацией и русско-язычный ресурс по unittest) протестировать возможные варианты работы программы (в том числе и возникновение исключительных ситуаций) для калькулятора.
4. Документировать функции *calculate*, *convert_precision*, *load_params* с помощью docstring. Включить в docstring тесты для функций, где это необходимо (см. пример и документацию).
5. Отрефакторить код таким образом, чтобы программа работала максимально стабильно, реагировала адекватно на ввод некорректных значений.

Опишем конкретные аспекты наиболее трудных заданий подробнее.

1. Анализ мест в коде с исключительными ситуациями
Исключительная ситуация может возникнуть на этапе работы с файлом (чтение, запись), обработки аргументов, вводимых пользователем, вычисления математических действий внутри функции calculate. Эти ситуации мы можем обработать с помощью блока (см. рабочий пример в стартовом борде):

```
try

    pass # какое-то выражение, возможно, поднимающее исключение

except Exception:
   print('Исключение возникло') # обработка исключения
else:
   # блок, выполняющийся, если исключения не было
```

2. Модульное тестирование с unitest

Шаблон для тестирования с помощью unittest может выглядеть так:

```
import unittest


class TestSomeFunc(unittest.TestCase): # создаем свой класс для тестов


     def firsttestcase(self): # внутри функции один или несколько тестовых
         self.assertEqual(2*2, 4) # случаев, которые проверяют какие-то
         # близкие предположения
```

```
    # ...
    def secondtestcase(self): # вторая группа тестов
        pass


unittest.main(verbosity=1)     # запуск тестов
```

Пример тестирования двух функций convert_precision и two_sum, которую мы создавали ранее. Нюанс тестирования в repl.it и PyCharm. В repl.it тесты запускаются вручную с помощью вкладки Shell (справа) (пример борда), в PyCharm требуется закомментировать запуск тестов с помощью:

```
unittest.main(verbosity=1)
```

3. Документирование docstring

Документирование - важный этап при написании программы почти любого масштаба. Документирование в Python осуществляется помимо обычных комментариев с помощью указания т.н. docstring с помощью многострочного варианта строки внутри функции. Приведем docstring для функций convert_precision. В приведенном примере сначала пишется краткое описание того, что делает функция, потом идут два примера вызова, которые также являются и тестами.

Лабораторная работа 5

Цель работы

Научиться считывать и записывать значения из файла и усовершенствовать калькулятор таким образом, чтобы было возможно конфигурировать его настройки (`PARAMS`) посредством файла, а также сохранять историю действий пользователя в файл.

Комментарии по выполнению

Работу можно разбить на две части:

1. Реализация функции загрузки параметров работы калькулятора из файла.

2. Реализация функции записи истории действий пользователя в файл.

Опишем каждую из них подробнее.

1. Реализация функции загрузки параметров работы калькулятора из файла

Эта функция подразумевает, что мы создадим вручную файл (допустим, params.ini) и напишем функцию, которая позволит считывать из него данные и присваивать считанные значения глобальной переменной PARAMS, объявленной в коде

```
def load_params(file="params.ini"):

    global PARAMS

    f = open(file, mode='r', errors='ignore')

    lines = f.readlines()

    for l in lines:

        print(l)
```

2. Реализация функции записи истории действий пользователя в файл

Пример работы программы

```
def write_log(file='calc-history.log.txt'):

    pass
```

[Стартовый борд в repl.it]

Лабораторная работа 6

Цель работы

Усовершенствовать приложение с калькулятором таким образом, чтобы оно позволяло:

выводить в красивом виде результаты вычисления операций на экран.

Комментарии по выполнению

Необходимо написать дополнительную функцию `print_results` таким образом, чтобы результаты вычисления выводились в "табличном" виде, границы таблицы оформляются с помощью символов `-, *, =`. Вывод должен быть организован в таком виде, чтобы таблица "подстраивалась" под любые введенные значения.

Пример №1

```
******************************
*   A   *   B   *    A + B    *
******************************
*  1234 * 12345 *   13579     *
******************************
```

Пример №2

```
**********************
*  A  *  B  *  A x B  *
**********************
*  2  *  5  *   10    *
**********************
```

Лабораторная работа 7. Тестирование unittest

Цель работы

Освоить принципы использования механизма обработки исключительных ситуаций при считывании/записи в файл на примере функции для сохранения лога операций и чтения настроек для работы калькулятора из файла.

Стартовый борд для задания

Комментарии по выполнению

Работу можно структурировать на следующие части:

1. Проанализировать ситуации работы с файлом, в которых может возникнуть исключение и реализовать обработку этих исключительных ситуаций с помощью базового инструментария, показанного в конспекте курса или по ссылкам (официальная документация, русско-язычный ресурс и ещё один по обработке исключений).
2. Применить принципы модульного тестирования и с использованием библиотеки unittest (см. пример в repl.it, сайт с официальной документацией и русско-язычный ресурс по unittest) протестировать возможные варианты работы программы по работе с файлом. Обратить внимание на возникновение исключительных ситуаций в этих операциях. Выделить ситуации при которых необходимо вручную поднять определенное исключение.
3. В стартовом борде рассмотрен способ тестирования поднятия исключения в случае, когда мы не используем специальных библиотек для считывания/записи в файл. Вам же нужно протестировать срабатывание исключений при использовании библиотек configparser (для чтения) и csv (для записи) файлов.

Опишем конкретные аспекты задания ниже.

1. Анализ мест в коде с исключительными ситуациями
Исключительная ситуация может возникнуть на этапе работы с файлом (чтение, запись). Программа может не считать файл с настройками например, из-за ограниченного набора прав

пользователя, запустившего данную программу или каких-либо настроек других ОС. Эти ситуации мы можем обработать с помощью блока (см. рабочий пример в стартовом борде):

```
try

    pass # какое-то выражение, возможно, поднимающее исключение

except Exception:
   print('Исключение возникло') # обработка исключения
else:
   # блок, выполняющийся, если исключения не было
```

2. Модульное тестирование с unitest

Шаблон для тестирования с помощью unittest может выглядеть так:

```
import unittest


class TestSomeFunc(unittest.TestCase): # создаем свой класс для тестов


     def firsttestcase(self): # внутри функции один или несколько тестовых
         self.assertEqual(2*2, 4) # случаев, которые проверяют какие-то
         # близкие предположения



     # ...
     def secondtestcase(self): # вторая группа тестов
         pass



unittest.main(verbosity=1)     # запуск тестов
```

Пример тестирования двух функций convert_precision и two_sum, которую мы создавали ранее. Нюанс тестирования в repl.it и PyCharm. В repl.it тесты запускаются вручную с помощью вкладки Shell (справа) (пример борда), в PyCharm требуется закомментировать запуск тестов с помощью:

```
Unittest.main(verbosity=1)
```

В итоге должна получиться полноценная программа «Калькулятор», собранная в одном репозитории