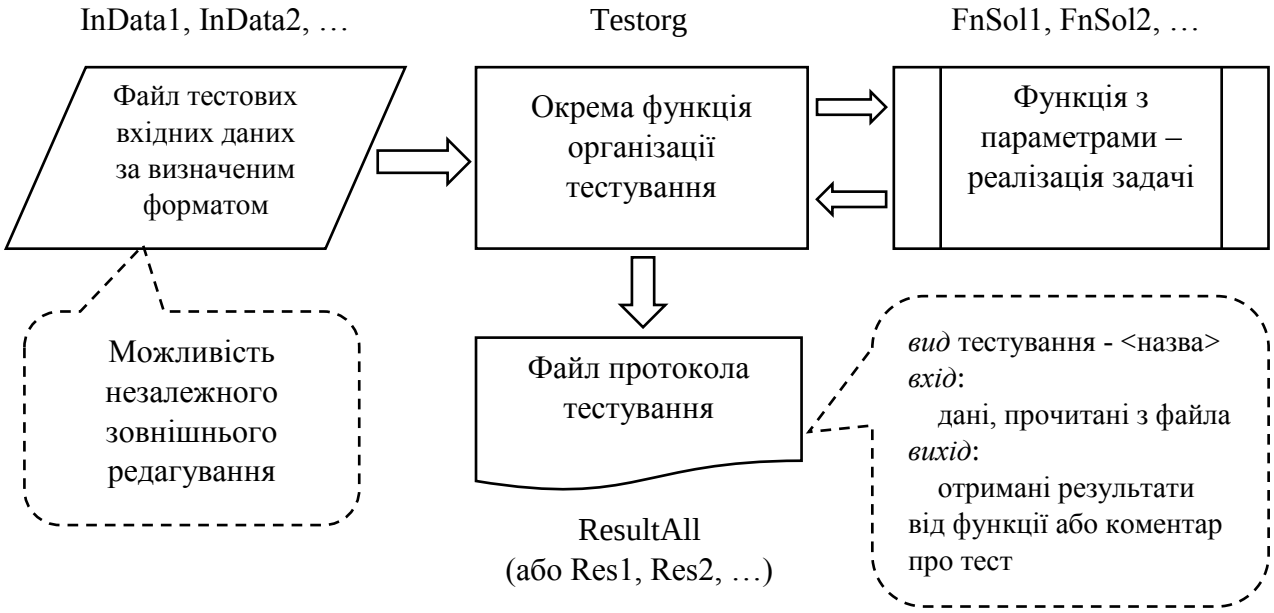
Це завдання пролягає у вивченні чи повторенні технології розробки і тестування функцій. Завдання треба реалізувати мовою Python, але зауважимо, що викладені засади є актуальними і для інших алгоритмічних мов.

Що треба виконати: 1)реалізувати функції відповідно до умов задач (вони є прості); 2)розробити тести для перевірки кожної функції; 3)організувати допоміжні функції і файли тестів; 4)виконати тестування і показати результати.

Спочатку визначимо архітектуру організації тестування. Отже, треба розробити тести за переліком критеріїв, викладених далі, і побудувати просту систему організації тестування. Не плутати терміни: *тест* – конкретний зміст вхідних даних і очікуваних результатів; *система тестування* – інструментальний засіб підготовки і виконання процедури тестування. Існує багато готових систем тестування, проте для першого етапу розробки достатньо і зручніше мати власний модуль тестування, що й пропонуємо виконати.



Умови задач

Задача 1. З файла читають цілі числа різних знаків. Послідовність чисел закінчується нулем, наприклад:
3 28 -4 901 666 -25 -25 700 -1 0
Обчислити: 1) кількість додатніх чисел; 2) кількість від'ємних чисел; 3) середнє арифметичне від'ємних.

Задача 2. Задано три числа *a, b, c* цілі або дійсні. Виначити: 1)чи можуть такі числа означати довжини відрізків; 2)якщо означають, то чи можна з таких відрізків утворити трикутник і який – рівносторонній, рівнобедрений, різносторонній.

Задача 3. Задане речення поділити на слова і надрукувати їх в стовпчик.

Задача 4. Задано двовимірний масив додатніх, від'ємних і нульових цілих чисел розміром N×N. Знайти підпрямокутник з найбільшою сумою. [Оптимізація і швидкість не потрібні.]
Підпрямокутник – це будь-який неперервний підмасив прямокутної форми заданого масиву.
Сума підпрямокутника – це сума всіх його елементів.

Запитання до кожної задачі: 1) чи умова задачі є точною і повною? 2) що треба уточнити в умові задачі?

Тестування - критерії

Насамперед зазначимо, що треба вивчити статтю про тестування за посиланням https://uk.wikipedia.org/wiki/Тестування_програмного_забезпечення

Розробити тести за такими показниками (див. статтю):

- Що оцінювати:
  - правильність відповіді для всіх можливих вхідних даних;
  - практичність.
- Використати метод тестування "чорної скриньки". Отже кожна функція має бути "закритою" для перегляду і редагування, зв'язок – лише через параметри і return.
- Організувати види тестування:
  - функціональне;
  - деструктивне;
  - зручності використання.
- Рівень тестування:
  - модульне;
  - системне в межах альфа-тестування.

Для кожної задачі скласти один файл тестових даних. [Файли можна об'єднати як буде сказано в кінці.] Кожний файл тестів повинний мати їх перелік відповідно до записаних критеріїв. Щоб правильно створити тестовий файл і правильно його читати, треба точно визначити його *формат* якою-небудь формалізованою системою, наприклад розширеною нотацією BNF.
[Пропонуємо окремо записати означення формата.]

Кодування

Скласти функції за умовами задач. Функції мають бути закритими від зовнішнього середовища, отже передавання даних до функції і отримання результатів реалізувати лише через параметри і return. Функції не мають використовувати побічні ефекти, наприклад, зв'язки з глобальними змінними.

Зберегти імена функцій і файлів як на малюнку.

Всі функції FnSol1, FnSol2, … оформити в одному модулі. В ньому ж визначити функцію Testorg організації тестування в розділі самотестування:

```
if __name__ == '__main__':
  # функція Testorg
  зробити запит на консоль про номер задачі
  зв'язати з файлом тестових вхідних даних
  отримати результати від функції
  зробити запис в файлі протокола тестування
```

Зважити, що перевірка робіт буде виконана вірогідно в базовому середовищі `Python's IDLE`. Отже, всі *налаштування програми і параметри компіляції* чи виконання, а також *підключення зовнішніх ресурсів* треба записати безпосередньо в тексті модуля, а не як налаштування власного середовища програмування для Python. [Такий підхід дозволяє інтегрувати програму до інших середовищ розробки]. Крім того, треба використати лише стандартні ресурси будови програм, наприклад, стандартні бібліотеки функцій. Нестандатні ресурси потребують окремого приєднання до програми (сценарію) і окремого передавання користувачам додаткових ресурсів.

Скласти *тести* для перевірки кожної задачі, як записано вище. Тести треба запускати циклом всі послідовно для кожної задачі.

Для цього завдання не варто надмірно ускладнювати програму, наприклад, засоби ООП тут не потрібні. Достатньо використати технологію процедурної (функціональної) *декомпозиції*.

В результаті виконання завдання надіслати у відповідь:

- один спільний модуль всіх функцій;
- файли InData тестових даних, чи об'єднаний файл; якщо файл об'єднаний, тоді функція Testorg має розпізнавати потрібні ділянки файла відповідно до задачі і формата файла;
- файл протокола тестування ResultAll, отриманого на власному комп'ютері, або окремі файли протоколів Res1, Res2, … ; не забувати додавати коментарі до файлів протоколів, як показано на малюнку; зміст протокола тестування має бути зрозумілим для експерта.