2020/2021 н.р.
Лабораторна робота 5 (2-й семестр) beta-версія

Структури даних та файли, або Text = 15 балів

Передмова

Мета

Метою лабораторної є обробка командного рядка, елементарна робота з файлами, знайомство з форматами csv та json, організація завантаження даних у внутрішні структури даних, а також нескладна статистична обробка даних та побудова власної системи класів.

Зрозуміло, що конкретно цю задачу можна розв'язати засобами баз даних. Зрозуміло, що можна побудувати більш ефективне за пам'яттю та часом роботи розв'язання. Але не завжди більш ефективний розв'язок буває економічно доцільним з точки зору вартості/часу його розробки.

Умова дуже коротко

Оброблювана інформація подається на вхід у двох текстових файлах.

Основна інформація подається в текстовому файлі, записаному у форматі **csv** (з роздільниками **;**) без заголовків полів. Рядки білих символів допускаються, помилками не вважаються і під час завантаження мають ігноруватись. Білі символи навколо роздільників не ігноруються.

До основного файлу додається допоміжний файл у форматі **json**, що містить підсумкову інформацію (у вигляді словника) щодо вмісту основного файлу.

Інформацію необхідно завантажити в пам’ять, перевірити її коректність. Далі необхідно видати певну статистику за отриманими даними.

Варіант задає:

1) порядок полів основного файлу;

2) ключі значень, що записано в допоміжному файлі;

3) специфічні обмеження на значення окремих полів;

4) яку інформацію, як та в якому порядку слід вивести в якості результату.

Детальні умови та розподіл варіантів наведено в окремих файлах.

Варто звернути увагу, що деякі обмеження, що випливають із здорового глузду, у варіантах можуть не вказуватись (наприклад, що вартість має узгоджуватись з кількістю та ціною, що ціни не можуть бути від'ємними, що мінімальне значення не може бути більшим максимального, тощо).

1 Програма повинна вміти

Завантажувати вміст текстових файлів з інформацією в пам’ять та перевіряти при цьому наявність синтаксичних та семантичних помилок у вхідних даних.

Знаходити та виводити певну статистику (згідно варіанту).

2 Вхідні дані програми

Усі вхідні дані надходять в програму з рядка виклику (= командного рядка). Під час роботи програми користувач не здійснює жодних введень.

Необхідна для роботи програми інформація записується в окремому json-файлі (в кодуванні utf-8), ім’я якого вказується в рядку виклику програми.

Програма має обробляти командний рядок такого вигляду.

<програма.py> <шлях до файлу з налаштуваннями>

У залежності від використовуваної ОС запуск програми на виконання буде здійснюватися так:

lab5.py task0.ini

або так

python-3.9 lab5.py task0.ini

тут **task0.ini** – файл з налаштуваннями. (Назви файлів значення не мають, тут наведено тільки для прикладу.)

Правильний json-файл з налаштуваннями містить словник[1] з ключами **input** та **output**. Файл записано в кодуванні utf-8 (без BOM).

Значення, що відповідає ключу **input**, є словником з ключами:

csv — шлях до csv-файлу з основною інформацією;

json — шлях до json-файлу з додатковою інформацією;

encoding — кодування вхідних файлів з інформацією.

Значення, що відповідає ключу **output**, є словником з ключами:

fname — шлях до файлу, в який необхідно вивести результат обробки;

encoding — кодування вихідного файлу.

Приклад оброблюваної інформації

Записи основного файлу містять поля: номер залікової книжки (рядок), навчальна група (рядок), порядковий номер листа з лабораторною роботою (ціле), код лабораторної роботи (рядок), кількість балів за спробу (ціле).

У допоміжному файлі наявні ключі: кількість записів основного файлу, сумарна кількість балів, SMILE[2].

Приклад основного файлу: **good.csv** .

Приклад допоміжного файлу: **good.json** .

Приклад файлу з налаштуваннями: **task0.ini** .

[1] в термінології json словникам Python відповідають об’єкти

[2] Не варто шукати прихований зміст у таких назвах ключів. Це банальна необхідність зробити 140 більш-менш різних варіантів ☹. Значення таких "дивних" ключів жодного значення не має. Але перевірка наявності ключа має бути.

3 Загальна схема роботи та консольний вихід програми

Виконання починається, як зазвичай, з виведення інформації про виконавця, номер варіанту та умову (тільки номер варіанту не підійде). Далі на окремому рядку виводиться рядок `*****`.

Подальший консольний вихід програми є протоколом її роботи.

Кожній операції завантаження, перевірки або виведення даних має відповідати один рядок. Перед початком виконання операції виводиться повідомлення про намір щось зробити. Далі розпочинається виконання операції. Якщо операцію було успішно виконано, то додруковується **OK**.

Спочатку завантажується вміст файлу з налаштуваннями. Його завантаження вважається успішним, якщо файл було успішно розібрано та в ньому присутні всі необхідні дані. За успішного завантаження має бути виведений рядок

ini <шлях до вхідного файлу> : OK

Після успішного завантаження налаштувань програма починає завантажувати вхідні дані: спочатку основні, потім додаткові. За успішних завантажень має бути виведений такий протокол.

input-csv <шлях до вхідного csv-файлу>: OK

input-json <шлях до вхідного json-файлу>: OK

Далі програма виконує перевірку відповідності **csv** та **json**-файлів. За її успішності у підсумку має бути виведено.

json?=csv: OK

Якщо вміст **csv** та **json**-файлів невідповідний, то в підсумку маємо побачити

json?=csv: UPS

Далі, незалежно від перевірки відповідності, програма виконує обробку даних та виводить отримані результати.

Перед виведенням аналогічно завантаженню спочатку виводиться повідомлення про наміри

output <шлях до вихідного файлу>:

потім розпочинається виведення, по його завершенню аналогічно введенням додруковується ознака успішності (**OK**). За успішного виведення загалом має бути виведено таке

output <шлях до вихідного файлу>: OK

Як мають оброблятися помилки, розглянуто в окремому розділі (див. далі).

Для сукупності файлів, розглянутої у прикладі, на консолі мало б бути таке.

```
щось про виконавця та умову (уточнюється згідно варіанту)...
*****
ini task0.ini: OK
input-csv good.csv: OK
input-json good.json: OK
json?=csv: OK
output result.txt: OK
```

Увага! Слова `*****`, **ini**, **input-csv**, **input-json**, **json?=csv**, **output**, **OK** є критично важливими для тестера. Виведені зайві символи кінця рядка тестеру можуть дуже сильно не сподобатись; так само як і відсутні там, де він їх очікує. При цьому під час обробки протоколу роботи програми до зайвих білих рядків та до зайвих білих символів тестер буде намагатись (ну, наскільки вистачить часу…) бути толерантним.

4 Шляхи виконання лабораторної роботи

Ця лабораторна робота має бути виконана з використанням об'єктно-орієнтованого підходу. Використання засобів керування базами даних (тобто баз даних) категорично заборонено.

Підхід №1 – використання ООП

Варіанти лабораторних робіт побудовано так, що інформацію можна зобразити у вигляді системи класів. Для розв’язання задачі необхідно побудувати низку класів, що в сукупності зберігають інформацію з файлу. Вхідна інформація розподіляється по об’єктах розроблених класів. Перевірка обмежень на вхідні дані є відповідальністю саме розроблених класів, що зберігають дані.

В якості прикладу візьмемо таку задачу.

Текстовий файл містить інформацію щодо виконання студентами лабораторних робіт у семестрі.

Записи файлу містять поля:

— номер залікової книжки (рядок з 6 цифр),
— навчальна група (рядок, не більше 6 символів),
— порядковий номер листа з лабораторною роботою (додатне ціле),
— код лабораторної роботи (рядок, від 1 до 7 символів),
— кількість балів за спробу (ціле від 0 до 10).
Допускаються білі рядки, які мають ігноруватися.
Студент однозначно ідентифікується заліковою книжкою. Жодний лист не містить дві лабораторні одночасно.
Скільки студентів отримали щонайменше 8 балів за кожну лабораторну, що виконували?

Нехай клас *Інформація* містить усі вхідні дані. Питання задачі стосується студентів. Тому об'єкт класу *Інформація* має зберігати контейнер, окремі елементи якого зберігають інформацію про студентів. Дані щодо окремого студента мають зберігатись в об'єкті класу *Студент*. Кожен студент повинен зберігати лабораторні роботи, які він здавав. Отже, в об'єкті студента має бути вкладений контейнер, які зберігає об'єкту класу *ЛабораторнаРобота*. Для кожної лабораторної роботи мають зберігатися спроби її здати (знов в якомусь контейнері). Отже, виникає контейнер, що зберігає окремі спроби — об'єкти класу *Спроба*. Бачимо, що цей підхід породжує трирівневу систему класів (*Студент*, *ЛабораторнаРобота*, *Спроба*), по якій розподіляється вхідна інформація. Щодо контейнерів, то можна використовувати вбудовані типи мови Python або, за бажанням, розробити власні класи (наприклад, *КонтейнерЛабораторнихРобіт*).

Варіанти лабораторних робіт побудовано так, що у більшості варіантів породжується переважно дворівнева система класів, але є поодинокі винятки в обидва боки. У більшості варіантів клас верхнього рівня зберігатиме те, чого стосується статистика.

Зауваження. Якщо виникне непереборне бажання, то класи-контейнери варто будувати на основі бібліотечних контейнерів мови Python з використанням композиції (використання успадкування з ймовірністю 99% призведе до можливості порушення відкритими методами класу цілісності даних). Крім безпосередньо зберігання даних власні контейнери можуть давати якийсь додатковий сервіс (дотримання унікальності елементів, що зберігаються, тощо) та зберігати підсумкову інформацію щодо даних, які в них зберігаються. Наприклад, результат найкращої спроби для контейнеру спроб. Наявність власних класів-контейнерів: а) не вимагається; б) кількість балів не збільшить; в) в умовах мови програмування Python та конкретного варіанту може виявитись надлишковим проектуванням.

Наявність кількох класів не є достатнім для того, щоб розв'язання було класифікованим як «*використання ООП*». Має бути розроблена система класів, за якої програма співпрацює виключно з класом *Інформація*. Решта класів є допоміжними та в ідеалі мають приховуватись від широкого загалу.

Використання цього підходу має певну специфіку завантаження даних.

— У класах *Інформація* та *Студент* (та інших, крім класу, об'єкти якого є елементами найнижчого рівня вкладеності) мають бути наявні відкриті методи, що додають інформацію приведену до відповідних скалярних або рядкових типів. Рядкові типи для числових або логічних полів не використовуються.

Наприклад,
додатиСтудента(номер-заліковки, навчальна-група),
додатиСтудентуЛабораторну(номер-заліковки, код_лабораторної),
додатиСтудентуЛабораторнуСпробу(номер-заліковки, код_лабораторної, номер-листа, бали).
При цьому поля, які за умовою є числовими, повинні задаватися значеннями числових типів.

Усі відкриті методи, до яких є доступ у клієнтів класів, в якості аргументів повинні приймати дані числових типів або рядки. (Інформація про внутрішній формат збереження інформації для клієнтів має бути недоступна. Такий підхід підвищує гнучкість коду, бо тоді клієнти не залежать від деталей реалізації.)

Методи, що приймають об'єкти типів *Студент*, *Лабораторна*, *Спроба*, тощо, не забороняються, але вони **не мають бути відкритими**. Якщо вони потрібні назовні, то для них **слід побудувати відповідні обгортки**. Наприклад, метод *знайти(Студент)* не повинен бути відкритим. Якщо він для чогось потрібний, то відкритим має бути метод *знайти(номер_заліковки)*. Останній конструює об'єкт студента та використовує далі невідкритий метод *_знайти(Студент)*.

У жодному класі, призначеному для збереження даних, не повинно бути методів, що додають інформацію, що задається як вміст усього файлу, або вхідний потік, або нерозібрана частина тексту, або якийсь інший контейнер з елементами інформації (наприклад, з полями).

Відповідальність за збереження вхідної інформації з основного файлу покладається на клас *Інформація*. За межами його об'єкту ці дані зберігатися не повинні.

5 Завантаження вхідних даних

Зображення вхідних даних має бути відділено від структур даних. Незалежно від використовуваного підходу, наявні в програмі класи, що зберігають основну інформацію, нічого не повинні знати про зовнішній формат збереження даних в оброблюваних текстових файлах. Клас *Інформація* нічого не повинен знати про додатковий файл (але його об'єкти можуть зберігати певну статистику). Дані з додаткового файлу мають використовуватися для перевірки інформації основного файлу, в об'єкт класу *Інформація* вони не завантажуються, а після виконання перевірки можуть бути вилучені з пам'яті.

Читання csv та json-файлів виконується стандартними бібліотечними засобами. Читання основного файлу з даними виконується рядками: кожен рядок читається та інформація з нього записується в пам'ять. Рядки файлу в пам'яті не зберігаються.

Вважати, що рядки файлу мають помірну довжину і їх безпечно завантажувати цілком.

За використання ООП-підходу логіку завантаження вхідного файлу варто виділити в окремий клас **Builder**, що знає до яких типів мають бути перетворені поля прочитаного рядка, а також куди їх слід записати (які методи класів викликати, щоб дані опинилися там, де мають).

Має бути наявна функція **load_data**, що виконує завантаження вмісту основного вхідного файлу в існуючий об'єкт класу *Інформація* «під ключ». Якщо в цьому об'єкті щось було, то його вміст на початку завантаження скидається. У випадку наявності помилок об'єкт має залишатися порожнім.

6 Збереження, обробка та виведення результуючої інформації

Завантажені вхідні дані мають залишатися в пам'яті до кінця роботи програми. Вилучення з них тієї частини, що не буде виводитись, є неприпустимим. Створення копій вхідних або проміжних даних з метою вилучення непотрібного або вибірки потрібного також є неприпустимим. Розстановка міток типу "це треба вивести" також є неприпустимою.

Виведення вихідної інформації має здійснюватися за один прохід. Перед виведенням допустимо один раз відсортувати. Виведення має відбуватися під час проходу по даних, накопичувати інформацію в проміжному буфері програми забороняється.

Формат виведення. Під час виведення в якості роздільників використовувати табуляції. Символ табуляції в кінці рядка є неприпустимим. Оскільки планується автоматичне тестування, то нічого непередбаченого умовою виводитися не повинно. Вихідний файл записується в кодуванні, що задається налаштуваннями програми.

7 Обробка помилкових та неочікуваних ситуацій

В усіх випадках обробка даних в програмі виконується до першої помилки.

За наявності помилки на окремому рядку виводиться повідомлення

******* program aborted *******

після нього, можливо, виводиться деяка діагностика, і програма завершує свою роботу.

8 Інше

*S1. Усі надіслані версії лабораторної не порушують принципи академічної доброчесності.

S2. Усі власні модулі/класи/функції/методи належно задокументовані. Для частин інтерфейсу використовуються рядки документації, для решти – коментарі.

*S3. На початку кожного файлу з кодом задокументовано його автора, який співпадає з студентом, що здає лабораторну роботу. Програма правильно виводить свого виконавця та варіант (з номером включно).

*S4. Відсутні надлишкові коментарі та коментарі, що дублюють зміст інструкцій або перекладають їх на природні мови. Коментарі та документація мають на 100% відповідати коду, який коментують; інакше наслідки будуть вкрай негативними.

S5. Гарно написана лабораторна не повинна складатися з одного ру-файла.

S6. Проектування коду виконано з дотриманням принципів ООП, функціонального проектування. Класи мають відповідати принципу інкапсуляції та підтримувати цілісність даних. Стиль програмування має відповідати принципам ООП. Код має захищати свої дані від несанкціонованих змін. Відкриті методи не повинні дозволяти зіпсувати дані та не повинні передбачати наявність перевірок перед викликом.

S7. Робота з ресурсами ведеться коректно, ресурси вчасно віддаються операційній системі. Робота з ресурсами є безпечною відносно появи винятків.

S8. Використовувані в програмі імена є змістовними, їх призначення зрозуміле з їх назви. Довгих імен слід уникати.

S9. Функції не повинні бути довгими, не повинні мати кілька обов'язків.

S10. Код не має бути заплутаним та захаращений перевірками. Використання механізму винятків має бути доречним та адекватним.

S11. Код не повинен містити дублювання та невикористовувані фрагменти (навіть закоментовані).

S12. Код має бути структурованим:

— * інструкція **break** можлива тільки всередині циклу **for**;

— * інструкція **return** всередині циклу є забороненою;

— * засоби дострокового завершення програми також заборонені для використання;

— використання винятків має бути доречним та не підміняти собою інші керуючі конструкції.

S13. Інструкції **try** не повинні бути вкладеними (за текстом) в інші інструкції **try**.

*S14. Інструкції **global**, **nonlocal** заборонені до використання;

S15. Код не містить «магічних» констант та власних глобальних змінних.

*S16. Код має відповідати рекомендаціям підрозділів 7.1.6 та 7.4 (див. файл **07 Керування порядком обчислень.pdf**).

*S17. Користувач не вводить жодної інформації. Навіть у кінці не натискає клавішу **<Enter>**.

*S18. Власні пакети прошу не використовувати, бо поточна версія тестера їх не подолає (а вільного часу на його модифікацію нема). Імпорт небібліотечних модулів під час тестування буде виконуватися з каталогу запуску програми (причини ті самі).

*S19. Імпортувати можна тільки це: власні модулі, модулі **csv**, **json**, **re**, **datetime** стандартної бібліотеки.

*S20. Усі ідентифікатори записані символами US-ASCII. Для файлів з текстом програми використовується виключно кодування **UTF-8** (без BOM).

*S21. Використання **async** не дозволяється. (Хоча це неймовірно, щоб в цій лабці воно комусь дійсно було потрібно.)

S22. Змінні, що використовуються в тілі функції виключно як локальні змінні функції для збереження проміжних результатів, не повинні бути її параметрами.

S23. Кожен фізичний рядок файлу з текстом програми цілком вміщується в одну ширину екрана ноутбука. (Текст програми можна прочитати в текстовому редакторі без горизонтальної прокрутки.)

9 Академічна порядність

Студент вільно орієнтується в коді лабораторної роботи, яку він здає, розуміє усі використані синтаксичні елементи мови та бібліотечні засоби, зміст та призначення частин коду, вміє самостійно запустити програму на виконання, здатен самостійно внести неглобальні виправлення в код. У коді відсутні коментарі, що майже буквально перекладають зміст інструкцій на природні мови.

Порушення пп. S3, S4 буде розглядатися як порушення принципів академічної порядності.

Принципи академічної доброчесності передбачають, що ані брати чужий код, ані давати комусь свій не можна. Сумісна розробка лабораторних також заборонена.

Якщо листи з кодом та скріншотами лабораторної відправлено не з власної поштової адреси (а з адреси товариша!), або якщо з однієї адреси відправлено лабораторні різних студентів, або якщо студент помилився в номері варіанту, або виконав не свій варіант, або неправильно зазначив виконавця, або виконавцем зазначений хтось інший, то все це також вважається порушенням академічної доброчесності.

Лабораторна робота успішно пройшла програмну перевірку на запозичення.

Лабораторна успішно пройшла співбесіду.

10 Оцінювання лабораторних робіт

Максимально можлива кількість балів за лабораторну становить 15 балів. Пороговий рівень, за якого лабораторна робота вважається успішно виконаною, складає 9 балів. За порушення принципів академічної порядності лабораторна робота оцінюється в 0 (нуль) балів.

Під час перевірки буде використовуватися Python-3.9.

Попередня програмна перевірка

Якщо буде час, то буде реалізована. Можливо, будуть зроблені тільки "димові" тести та ще невелика частина; можливо, буде повнофункціональна перевірка.

Розраховувати на те, що всі критичні для отримання позитивного результату вимоги будуть перевірятись заздалегідь та програмно не варто! Іноді тестер може давати підказки щодо потенційних порушень (можливо, що порушення є, можливо, що нема). Їх варто прочитати.

Звертаю увагу, що є частина вимог, пов'язана з оформленням і через порушення яких тестер не зможе обробити суть та буде писати на перший погляд дивні речі. Тут 1) **тестер правий, що відхиляє лабораторну, бо він працює згідно загальних вимог, які зазначені в цьому файлі**; 2) варто перевіряти, чи в тому вигляді, послідовності, форматі виводиться те, що має виводитись.

Щодо того, що попередньо перевірятись не буде: не найгірша думка самостійно протестувати власний код згідно вимог. Також є викладачі, що ведуть лабораторні заняття. Не найгірша думка інколи з ними консультуватись.

Якщо раптом виявляється, що код стає заплутаним, у ньому з'являються різноманітні "затички", збільшується кількість перевірок та розгалужень, то це явна ознака того, що щось явно не так.

Перевірка лабораторних робіт

Офіційна перевірка лабораторних робіт розпочнеться ПІСЛЯ завершення терміну приймання.

Розв’язувана задача має відповідати умові та варіанту.

Отримана сукупність лабораторних робіт ПРОГРАМНО перевіряється на дотримання принципів академічної доброчесності (запозичення коду). Чим більше буде збіг, тим більш ретельною буде співбесіда. Якщо збіг буде зашкалювати, то всі такі лабораторні будуть оцінені в 0 балів, незалежно від того, хто був клієнтом, а хто сервером.

Не таємниця, що умови майже однакові і на лабораторних заняттях схожі задачі розв'язуються. Але власноруч набраний код, навіть за мотивами розібраної схожої задачі, буде набагато більш різноманітним, ніж код, отриманий заміною деяких символів у набраному кимось файлі. І програма це чудово вміє враховувати, щоб визначити код, який був переважно просто скопійований.

Лабораторна робота передбачає захист у формі співбесіди (письмова + усна частини). Письмова частина обов'язкова для всіх.

Співбесіда за лабораторною (усна частина) може відбуватися не тотально, а тільки в тих випадках, коли наявна інформація (інші результати оцінювання, надісланий код та скріншоти, відгуки викладачів, що ведуть лабораторні заняття; результати програмної перевірки на запозичення) або її відсутність ставить під сумнів дотримання принципів академічної доброчесності під час виконання лабораторної роботи.

Захист лабораторної роботи вважається неуспішним, якщо під час захисту виявляється, що студент не до кінця розуміє код або погано в ньому орієнтується чи не розуміє використані синтаксичні елементи мови, зміст та призначення частин коду, а також якщо захист не відбувся з ініціативи студента.

IF (тестер відхилив лабораторну,
або розв'язувана задача не відповідає умові чи варіанту,
або хоча б одна версія порушує принципи академічної доброчесності,
або лабораторна має критичну кількість запозичень,
або лабораторна порушує формат вихідних даних (протокол роботи, що виводиться на консоль (див. підрозділ 3), або формат даних, що виводяться у файл (див. підрозділ 6)),
або лабораторна порушує вимоги підрозділу 6 або якийсь з пунктів, відмічених *,
або на коректних вхідних даних видає неправильний (за змістом або за формою) результат,
або захист був неуспішним) :
лабораторна отримує **0** балів
ELIF наявні порушення вимог хоча б одного з підрозділів 4, 5, 6:
лабораторна отримує **9** балів
ELSE:
кількість_балів = 9
IF лабораторна правильно обробляє некоректні вхідні дані:
кількість_балів += 3
IF лабораторна відповідає всім вимогам (і нумерованим, і ненумерованим,
але не пов'язаним з некоректними вхідними даними), зазначеним у цьому файлі:
кількість_балів += 3
лабораторна отримує **кількість_балів** балів

Порада. Перед тим, як надсилати код лабораторної, варто її ретельно самостійно протестувати, а також з олівцем перевірити виконання вимог (методом: перевірили, якщо виконується, поставили галочку). За необхідністю внести зміни і ще раз протестувати. І якщо все гаразд, то лабораторну можна здавати ☺.

Організаційні моменти

Не пізніше 19 травня 2021 року на адресу LabAssignment2@i.ua (далі «адреса для лабораторних робіт») надійшов лист з повним кодом лабораторної роботи (усі суттєві для проекту ру-файли і нічого іншого, як вкладення). У темі листа зазначено, що це лабораторна робота 5 та номер варіанту, у форматі **Lab5, <номер варіанту>**. Наприклад, **Lab5, 66**

У самому листі зазначено виконавця та середовище, що використовувалося для виконання лабораторної роботи. Листи з архівами та посиланнями на інтернет-ресурси не припустимі. Один лист – одна лабораторна робота, повністю.

Дозволяється відправляти код лабораторної кілька разів (наприклад, якщо було усунуто якісь недоліки). У випадку, коли код лабораторної надходив кілька разів, розглядатиметься та оцінюється тільки остання версія (навіть, якщо передостання працювала, а до останньої забули додати частину файлів). Дата/час версії визначається за датою надходження на адресу для лабораторних робіт. Кількість спроб на оцінювання не впливає.

Орієнтовно. Письмова частина співбесіди за лабораторною роботою відбуватиметься орієнтовно 25-27 травня (одночасно з модульною контрольною). Усна частина вже потім, по групах/підгрупах. Або того ж тижня, або наступного. Залежно від кількості учасників (сподіваюсь, що масовості з усною частиною не виникне).

Підказки до лабораторної роботи

Рядок виклику та його обробка – див. розділ 16.

Для тих, хто збирається чесно підійти до лабораторної з позицій ООП, у цьому файлі буде кілька підказок (і не тільки для них). Крім того наявна друкована інформація, а саме.

У методичці *Вступ до програмування мовою С++. Том 3. Структури даних*, наявна також електронна https://drive.google.com/open?id=1ZE67gBlZJSBSP7KuoJUZwgc_Z-YHKTWx є глава 5 *Задача про обробку тексту*. У цій главі розглядається розв'язання задачі, дуже схожої на задачу лабораторної роботи. При цьому розв'язання будується в ООП-стилі. Те, що мова програмування інша, не змінює загальної структури проекту. Тому варто подивитись у текст, не звертаючи уваги на синтаксичні відмінності. Імена функцій та методів там було взято змістовні, тому здогадатись, що виконує функція, можна й без розуміння синтаксису.

Поточна лабораторна відносно описуваної у методичці має низку суттєвих спрощень:

1) лексичний розбір рядка виконується бібліотечними засобами (csv);

2) усі контейнери беруться як вбудовані типи Python (ні, щось своє не забороняється; тільки це може стати надлишковим проектуванням).

Лабораторна робота не передбачає деталізовану діагностику помилок. Тому в прикладі вона проектується "економно": тільки те, що вимагається в умові (без жодної деталізації діагностики помилок та жодних підказок користувачу щодо командного рядка та структури файлу з налаштуваннями).

Щодо використання ООП-підходу до організації збереження даних. Може виявитись набагато простіше все-таки зробити систему класів, що зберігають інформацію, ніж намагатись все зберігати в табличному вигляді в одній структурі даних. Бо в розділі 6 є багато вимог, виконати які з табличним зображенням даних буде доволі складно; у деяких варіантах навіть набагато складніше за розробку системи класів.

Порядок виконання лабораторної роботи

Під час виконання лабораторної найголовніше – **не** потрапити в ситуацію:
"Ця робота не може бути завершена, бо її було неправильно розпочато".

Виконання лабораторної роботи має сенс розбити на окремі етапи, які розглядатимуться далі.

Як приклад для ООП-підходу буде розглядатися задача.

Текстовий файл, записаний у форматі csv, містить інформацію щодо виконання студентами лабораторних робіт у семестрі.

Записи файлу містять поля:

— номер залікової книжки (рядок з 6 цифр),

— навчальна група (рядок, не більше 6 символів),

— порядковий номер листа з лабораторною роботою (додатне ціле),

— код лабораторної роботи (рядок, від 1 до 7 символів),

— кількість балів за спробу (ціле від 0 до 10).

Допускаються білі рядки, які мають ігноруватися.

Студент однозначно ідентифікується заліковою книжкою. Жодний лист не містить дві лабораторні одночасно.

До цього файлу додається додатковий файл, що містить інформацію:

— кількість записів основного файлу (ціле),

— сумарна кількість балів (ціле).

Додатковий файл записано у форматі json.

У цій задачі лабораторну вважати виконаною, якщо за неї отримано 6 або більше балів.

Балом, отриманим за лабораторну, вважати кількість балів найкращої спроби.

Знайти всіх студентів, що отримали щонайменш 8 балів за кожну лабораторну, що виконували. Вивести по кожному з них інформацію:
на першому рядку:
номер залікової книжки, група, загальна кількість листів з лабораторними
на наступних рядках, починаючи з табуляції, вивести по кожній зданій студентом лабораторній (по одній на рядок):
код лабораторної, номер листа з останньою найкращою спробою, кількість балів за спробу
у сортуванні: кількість балів за спробу (за спаданням), код лабораторної.

1 Знайомство з умовою варіанту

Зміст етапу. З'ясувати які дані треба обробляти, що має бути на виході, підготувати тестові дані.

1A Уважно прочитати умову лабораторної в частині варіанту. З'ясувати які дані має обробляти програма, що вона має виводити.

1B Побудувати приклади вхідних даних:

— приклад файлу з основною інформацією; не дуже великий, але не зовсім тривіальний; рядків 10-20-30 буде достатньо;

— до файлу з основною інформацією побудувати додатковий файл.

1C Для побудованих вхідних даних побудувати (вручну) текстовий файл, який має бути отриманий на виході.

1D Уважно прочитати в умові варіанту, в якому порядку мають розташовуватися рядки у вихідному файлі. Перебудувати файл, отриманий на кроці 1C, щоб в ньому було належне сортування.

Надалі отримані файли можна буде використовувати в якості тестових даних.

2 Загальна структура програми та обробка командного рядка

Зміст етапу. Сформулювати загальну архітектуру програми. Реалізувати обробку командного рядка згідно вимог лабораторної роботи. Започаткувати обробку помилок.

У результаті має бути отриманий працюючий код, який як тільки будуть належно реалізовані його складові, почне працювати.

Підготовка.

Передивитись *16 Рядок виклику та обробка аргументів виклику.pdf*.

Подивитись як виглядає файл **task0.ini**.

У частині *2 Вхідні дані програми* умови лабораторної подивитись, як має виглядати рядок виклику програми.

Створити власний файл з налаштуваннями, що задає обробку створених на попередньому кроці основного та додаткового файлів. В якості вихідного файлу вказати ім'я файлу, що **відрізняється** від файлу, побудованого на попередньому кроці! (Той файл є еталонним. Перезаписувати його не треба.)

Загальна структура програми

0 Виведення загальної інформації

0A Вивести інформацію про виконавця задачі

0B Вивести варіант та умову задачі

0C Вивести рядок *****

1 Обробити командний рядок (у т.ч. див. 7 Обробка помилкових на неочікуваних ситуацій)

Якщо кількість аргументів неналежна, то гарантувати виведення повідомлення

******* program aborted *******

далі (за бажанням вивести довідку з використання програми) та завершити виконання.

(Ось тут про try-except варто не згадувати. If-else для обробки командного рядка буде більш ніж достатньо.)

За належної кількості аргументів викликати функцію, що отримує на вхід ім'я файлу з налаштуваннями та виконує всю роботу «під ключ».

З наведеної структури випливає, що мають бути:

1) функція виведення інформації про виконавця, причому ця функція нічого не знає ані про умову, ані про рядок *****;

2) функція виведення умови, бо вона довга; причому ця функція нічого не знає ані про виконавця, ані про рядок *****;

3) функція, що за іменем файлу з налаштуваннями виконує всю роботу під ключ (надалі **основна** функція).

Що можна закодувати зараз:

1) загальний алгоритм;

2) функцію виведення інформації про виконавця;

3) функцію виведення варіанту та умови;

4) заглушку основної функції.

Написати остаточний код основної функції зараз не вдасться, бо ще багато чого треба закодувати. Але написати «заглушку», яка прикидається, що щось виконує – запросто.

На даний момент буде достатньо, щоб основна функція приймала один аргумент та виводила повідомлення «я функція така-то, мені дали такий-то аргумент».

Для чого? Щоб протестувати як закодовано загальний алгоритм.

Що робимо далі: тестуємо роботу існуючої програми.

Треба перевірити:

Як працює програма, коли в командному рядку записані зайві аргументи.

Як працює програма, коли в командному рядку записано недостатньо аргументів.

Як працює програма, коли в командному рядку записана правильна кількість аргументів.

Якщо виникли негаразди – виправляємо та тестуємо знов. Процес повторюємо, поки не запрацює як слід.

3 Уточнення основної функції

Починаємо уточнювати основну функцію, скажімо **process**. Вона повинна прочитати файл з налаштуваннями та виконати обробку інформації.

Повертаємось до умови лабораторної:

Цитата

Кожній операції завантаження, перевірки або виведення даних має відповідати один рядок. Перед початком виконання операції виводиться повідомлення про намір щось зробити. Далі розпочинається виконання операції. Якщо операцію було успішно виконано, то додруковується **OK**.

Цитата

В усіх випадках обробка даних в програмі виконується до першої помилки.

За наявності помилки на окремому рядку виводиться повідомлення

******* program aborted *******

Після нього, можливо, деяка діагностика (див. далі), і програма завершує свою роботу.

Наведені вище вимоги можна виконати, якщо побудувати обробку помилок з використанням винятків:

— Виводиться повідомлення про намір щось зробити.

— Виконується запланована дія.

— Якщо вона неуспішна, то десь генерується виняток і виконання передається пастці, в якій друкуються необхідні повідомлення про помилку.

— Якщо все гаразд (тобто винятку не було), то додруковуємо **OK**.

(При цьому всі виконувані основною функцією дії з даними мають стояти під спільним **try**.)

Висновок: тіло **process** має огортатись у **try**, з **except**-секції якої виводиться повідомлення

******* program aborted *******

Якщо деталізована діагностика помилок не передбачається, то це може виявитись єдиною інструкцією **try** у програмі.

Виконання кожної запитуваної дії (завантаження налаштувань, даних, виведення, тощо) має виконуватися окремою функцією (у деяких випадках методом), яка за виникнення проблем генерує виняток.

Тепер розбираємось з окремими діями (див. частину *3 Загальна схема роботи та консольний вихід програми* умови лабораторної). На правильних вхідних даних має бути виведений такий протокол.

ini <шлях до вхідного файлу> : OK
input-csv <шлях до вхідного csv-файлу>: OK
input-json <шлях до вхідного json-файлу>: OK
json?=csv: OK
output <шлях до вихідного файлу>: OK

Цей протокол визначає загальні кроки, які має виконувати головна функція.
завантажити файл з налаштуваннями
ini < шлях до вхідного файлу> : OK
завантажити вхідні дані
input-csv < шлях до вхідного csv-файлу>: OK
input-json < шлях до вхідного json-файлу>: OK
json?=csv: OK
вивести запитувану інформацію
output < шлях до вихідного файлу>: OK

Завантаження файлу з налаштуваннями має виконувати окрема функція, скажімо, **load_ini**.
Вона отримує:
шлях до файлу з налаштуваннями
та:
відкриває файл з налаштуваннями
завантажує його вміст у пам'ять (необхідні засоби наявні в модулі **json**)
зараз наявний словник із налаштуваннями
перевіряє (див. *2 Вхідні дані програми*), чи наявні в словнику необхідні параметри
(*примітка*: наявність зайвих параметрів не є помилкою вхідних даних)
У випадку некоректного вмісту файлу з налаштуваннями має бути згенерований виняток (який буде перехоплений у функції **process**).

Завантаження вхідних даних має виконувати окрема функція, скажімо, **load** (в окремому *ру-файлі*). Вона має отримати як аргументи:
— *об'єкт класу *Інформація*, в який слід завантажити дані;
— ім'я основного файлу, який слід завантажувати;
— ім'я додаткового файлу;
— кодування, в якому записано файли.
Далі вона
— завантажує основний файл;
— завантажує додатковий файл;
— перевіряє відповідність інформації основного та додаткового файлів.
Кожна з цих дій реалізується окремою функцією.
* Замість того, щоб отримувати об'єкт, вона може його створювати. Вибір за виконавцем лабораторної.

За умовою лабораторної завантаження основного вхідного файлу має виконуватися функцією **load_data**, що виконує завантаження вмісту основного вхідного файлу в існуючий об'єкт класу *Інформація* «під ключ». Перед початком завантаження вміст об'єкту має скидатися і, якщо завантажити основний файл не вдалося, то об'єкт має залишатися порожнім, — ці дії мають виконуватися функцією **load_data**.

Отже, створюємо клас *Інформація* (*в окремому ру-файлі*). Поки в ньому є тільки один метод:
— метод **clear**, який скидає вміст об'єкта.

На поточний момент цей метод реалізується як порожній (тіло складається з єдиної інструкції **pass**).

Функція **load_data** (розміщується в *ру-файлі*, де **load**) має отримати як аргументи:
— об'єкт класу *Інформація*, в який слід завантажити дані;
— ім'я основного файлу, який слід завантажувати;
— кодування, в якому записано файл.
Поки вона не дуже знає, що робити з ними далі. Отже, на початку скидає поточний вміст об'єкту інформації.
Обробка помилок завантаження основного файлу.
Необхідно забезпечити, щоб за наявності помилок під час обробки об'єкт інформації скидався. Це можна зробити дуже різними способами. У тому числі реалізувати клас *Інформація* як контекстний менеджер, що за виходу за винятком буде скидати вміст об'єкта. *Hint*: метод **__exit__** контекстного менеджера приймає аргументи. Які будуть їх значення, якщо завершення виконання інструкції **with** відбулось у звичайний спосіб? (Інші способи організації скидання вмісту об'єкта за неуспішного завантаження також можливі.)

Функція, що завантажує вміст додаткового файлу, скажімо, **load_stat** (розміщується в *ру-файлі*, де **load**) має отримати як аргументи:
— ім'я додаткового файлу, який слід завантажувати;
— кодування, в якому записано файл.
Що виконує:
— завантажує додатковий файл;
— перевіряє наявність необхідних ключів (див. варіант);
— якщо все гаразд, то повертає прочитані з допоміжного файлу значення ключів.
Примітка. Вона може повертати словник, тільки інший – назви ключів варто змінити на внутрішні (коротші та, можливо, англійські). Чому: бо далі треба виконувати перевірку відповідності вмісту основного та додаткового файлів, і буде дуже погано, якщо функція, що перевіряє відповідність буде повинна знати про те, якою мовою та якими словами записано ключі.

Функція, скажімо **fit**, що перевіряє відповідність вмісту основного та додаткового файлів, винятки генерувати не повинна. Її задача повернути ознаку істинності (**True** або **False**). Вона має отримати об'єкт з інформацією та отриману з функції **load_stat** інформацію. Далі зробити якісь (поки магічні) дії та повернути результат перевірки. На поточний момент її варто зробити такою, що завжди повертає **True** (у Python ≥ 3.10 спроба оцінити істиннісне значення для **NotImplemented** може завершитись винятком).

Виведення запитуваної інформації можна зробити або окремою функцією, або методом класу *Інформація*. В ідеалі клас *Інформація* та не тільки він мав би підтримувати протокол ітерації (по ньому мало б бути можливим ітерування). За розвиненої роботи з ітераторами це беззаперечно мала б бути окрема функція, що використовує відкритий інтерфейс класу. Лабораторна робота такого жаху від виконавців не вимагає, тому цілком підійде більш простий (і не дуже правильний) варіант, коли виведення реалізується методом класу *Інформація*.
Тому додаємо до класу *Інформація* метод, скажімо, **output**. Цей метод має отримати аргументи:
— шлях до вихідного файлу;
— кодування, в якому слід записати вихідний файл.
Поки він не до кінця знає, що йому робити, тому може прикидатись, що гарно виконує свою роботу.

Що можна закодувати зараз:
1) започаткувати клас *Інформація*, додавши до нього методи **clear**, **output**.
2) функцію **load_ini**, що завантажує файл з налаштуваннями;
3) функцію **load_data**, що завантажує основний файл;
4) функцію **load_stat**, що завантажує додатковий файл;
5) функцію **fit**, що перевіряє відповідність вмісту основного та додаткового файлів;
6) функцію **load**, що завантажує основний, додатковий файл та перевіряє відповідність їх вмісту.
7) повністю зібрати функцію **process**.

Має бути 3 файли:
1) програма, основна функція;
2) все, що стосується завантаження;
3) клас *Інформація*.

Ступінь завершеності:
clear – поки заглушка;
load_data, **output** – започатковані і більше нічого;
load_ini, **load_stat** – остаточні версії;
fit – поки заглушка, завжди повертає **True**;
load – остаточна версія (усі складові в неї є);
process – остаточна версія (усі складові в неї є).

Що варто протестувати.
1. Програму в цілому (на коректних даних, вони були побудовані раніше), щоб переконатись у правильності протоколу, що виводиться.
2. Окремо функцію **load_ini** (обов'язково подивитись на те, що вона повертає).
3. Окремо функцію **load_stat** (обов'язково подивитись на те, що вона повертає; звернути увагу на кодування).
4. Якщо планується отримання балів за обробку помилок, то протестувати програму на даних, некоректних у частині файлу з налаштуваннями та/або додаткового файлу.

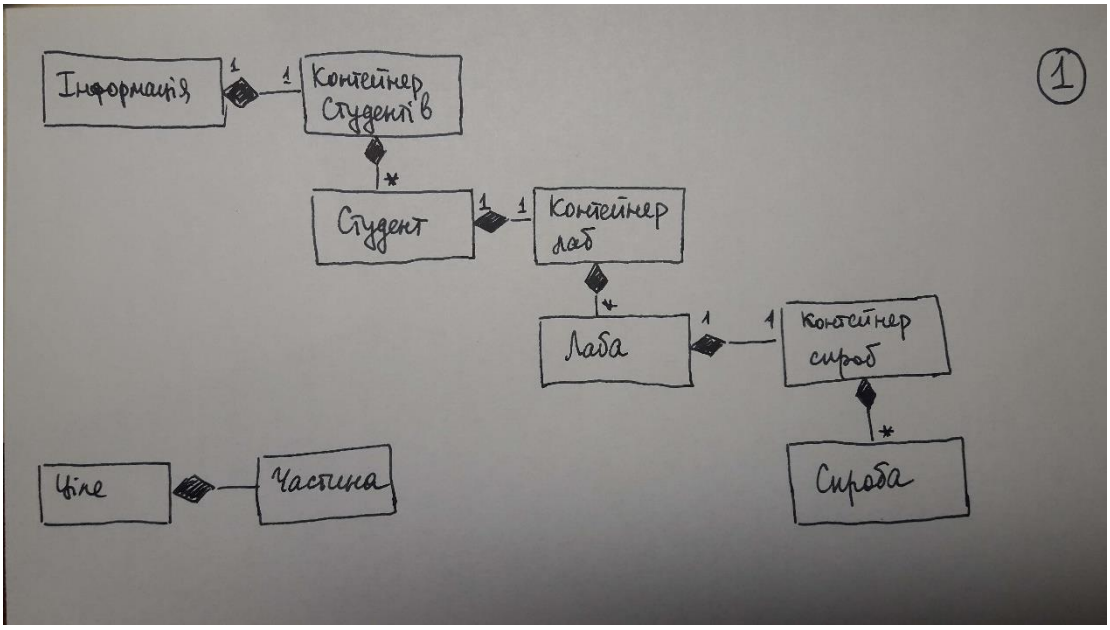
4 Уточнення членів-даних класу Інформація та супутніх класів

Ця частина стосується тільки підходу №1 виконання лабораторної (використання ООП), див. частину *4 Шляхи виконання лабораторної роботи* умови лабораторної.

На цьому етапі слід уточнити систему класів, що будуть зберігати наявну інформацію. Надалі розглядається побудова структури даних з використанням композиції, а не успадкування.

Система класів буде суттєво залежати від тієї інформації, яка має бути виведена. Згадаємо питання задачі:
Знайти всіх студентів, що отримали щонайменше 8 балів за кожну лабораторну, що виконували. Вивести по кожному з них інформацію:
на першому рядку:
номер залікової книжки, група, загальна кількість листів з лабораторними
на наступних рядках, починаючи з табуляції, вивести по кожній зданій студентом лабораторній (по одній на рядок):
код лабораторної, номер листа з останньою найкращою спробою, кількість балів за спробу
у сортуванні: кількість балів за спробу (за спаданням), код лабораторної.

Звідси випливає, що *Інформація* має керувати переліком *Студентів*. *Студент* має керувати переліком *Лабораторних*. *Лабораторна* має керувати переліком *Спроб*.



Класи *Інформація*, *Студент*, *Лаба* не повинні бути суто контейнерами (тобто ототожнювати їх з *КонтейнерСтудентів*, *КонтейнерЛаб*, *КонтейнерСпроб* не слід). Крім самого об'єкта-контейнера, вони можуть містити й іншу інформацію. Так, *Інформація* може зберігати якусь статистику (наприклад, загальну кількість виконаних студентами лабораторних). *Студент* крім лабораторних може зберігати номер залікової книжки та групу. Лабораторна крім переліку спроб та свого коду може «знати», чи успішно вона здана, з яким балом, в якому листі знаходиться «залікова» спроба, тощо (це робиться для того, щоб оптимізувати отримання підсумкової інформації).

Щодо контейнерів, то можна обійтись без окремих класів, використавши вбудовані контейнери мови Python: list, set, dict.

Слід уважно прочитати умову власного варіанту задачі, виписати в стовпчик усі поля, що наявні у вхідних файлах. Далі додати до них ті поля, які мають виводитись, але відсутні в явному вигляді у вхідному файлі (наприклад, загальна кількість листів з лабораторними). Далі додати поля, за якими треба буде сортувати. Далі подивитися критерії, за якими слід вибирати студентів, і додати відповідні поля.

Для прикладу це буде:
за основним вхідним файлом
— номер залікової книжки,
— навчальна група,
— порядковий номер листа з лабораторною роботою,
— код лабораторної роботи,
— кількість балів за спробу,
за додатковим файлом
— кількість записів основного файлу
— сумарна кількість балів
за тим, що треба виводити
— загальна кількість листів з лабораторними (для студента)
— номер листа з останньою найкращою спробою лабораторної (для студента)
— результат лабораторної (для студента)
за тим, що впливає на сортування
(нічого нового не додається)
за критерієм відбору студентів для виведення
— результат лабораторної (для студента) (вже було раніше)

Далі треба визначити де що буде зберігатися і чи треба зберігати. Концепція роботи з даними передбачає, що для завантаження та вивантаження має використовуватися однаковий формат. Саме через це в перелік потрапляє те, що зазначається в додатковому файлі. Поля, визначені за додатковим файлом, можуть або обчислюватися за об'єктом інформації, або зберігатися в ньому.

До *Інформації* потрапляють:
— кількість записів основного файлу *
— сумарна кількість балів*
До *Студента* потрапляють:
— номер залікової книжки,
— навчальна група,
— загальна кількість листів з лабораторними (для студента)*
— кількість зданих лабораторних (для студента)*
До *Лабораторної* потрапляють:
— код лабораторної роботи
— результат лабораторної (краща спроба)*
— номер листа з останньою найкращою спробою (краща спроба)*
(останні дві характеристики можна задати посиланням на відповідний об'єкт класу *Спроба*)
До *Спроби* потрапляють:
— порядковий номер листа з лабораторною роботою
— кількість балів за спробу

* може бути або безпосередньо зберігатися в об’єкті, або обчислюватися.

Примітка. Статистичну інформацію, яка отримується за вмістом контейнера в цілому, і за якою далі буде треба сортувати, обчислюваним атрибутом робити не слід. Бо тоді під час виведення буде виконуватись далеко не один прохід по даним і критичним чином буде порушено умову лабораторної (а лабораторна буде працювати так довго, що можна буде не тільки каву зготувати, але й банкет). Тобто те, чого стосується перший рядок виведення (про студента), з ймовірністю 99.9% має зберігатися або легко обчислюватися (тобто не потребувати під час виведення обходу даних) в об’єкті класу *Студент*.

Оскільки в лабораторній пропонується використання бібліотечних контейнерів, то далі варто вирішити які використовувати. Оскільки дані треба буде сортувати, то усі контейнери робимо типом **list**. Це у деяких варіантах може дати квадратичну оцінку часу завантаження, але суттєво зекономить пам’ять та спростить код, адже список можна відсортувати на місці.

За використання **set** під час виведення виникне потреба кожен раз утворювати відсортовану копію даних. Можливо, десь і **dict** підійде. Це залежатиме і від конкретної задачі, і від реалізації решти її частин.

Отже, у найпростішому випадку всі контейнери робимо типом **list**. Тобто класи *КонтейнерСтудентів, КонтейнерЛаб, КонтейнерСпроб* насправді є класом **list**. (Зробити їх такими, що зберігають список та його підсумкову інформацію, також можливо, але до такого поскладнювати систему класів не будемо.)

Далі можна писати класи *Інформація, Студент, Лаба, Спроба* у частині їх конструкторів. Не забуваючи в конструкторах класів *Інформація, Студент, Лаба* створювати порожній список, який буде списком студентів, списком лаб, списком спроб відповідно.

Тепер можна реалізувати метод **clear**, що скидає дані: він має не тільки спустошувати список, але ще й занулювати поля, що містять статистичну інформацію (за наявності).

Увага! У лабораторних роботах кількість рівнів вкладеності може бути іншою.

5 Уточнення методів класу Інформація та супутніх класів: додавання даних

Ця частина стосується тільки підходу №1 виконання лабораторної (використання ООП), див. частину *4 Шляхи виконання лабораторної роботи* умови лабораторної.

Перш за все, до класів слід додати методи, що добувають значення полів, що не є контейнерами, та, можливо, деяких обчислюваних атрибутів (property підійдуть). Геттери, що повертають контейнери, додавати категорично заборонено.

З одного боку, основний файл складається з рядків з такими полями
— номер залікової книжки (рядок з 6 цифр), **npass**
— навчальна група (рядок, не більше 6 символів), **ngroup**
— порядковий номер листа з лабораторною роботою (додатне ціле), **num**
— код лабораторної роботи (рядок, від 1 до 7 символів), **lr**
— кількість балів за спробу (ціле від 0 до 10), **points**

(праворуч вказано позначки, під якими вони будуть фігурувати далі)

З іншого боку, є клас *Інформація*, в який слід вміти додавати дані.

Додамо до класів методи, що будуть записувати в об'єкт дані. Починаємо згори.

У класі Інформація має бути метод

```
def load(npass:str, ngroup:str, num:int, lr:str, points:int)
```

що має розмістити в пам'яті дані.

Що він має робити:

за номером залікової книжки знайти студента,

якщо нема, то додати, якщо є, то перевірити його номер групи

записати дані про **num**, **lr**, **points** в об'єкт студента

оновити статистичну інформацію в об'єкті (збільшити кількість записів на **1**, сумарну кількість балів на **points**)

Звідси бачимо, що

— в *Інформацію* треба додати метод, що шукає студента за номером залікової книжки;

```
def find(npass:str)->Студент
```

— в *Інформацію* треба додати метод, що додає студента;

```
def add(npass:str, ngroup:str)->Студент
```

— у *Студента* треба додати метод, що записує в нього дані;

```
def load(num:int, lr:str, points:int)
```

Додавання даних студенту виконується аналогічно:

знайти лабораторну

додати в лабораторну дані спроби

оновити статистичні дані (кількість листів студента, кількість зданих лабораторних)

— у *Студента* додаємо метод, що шукає лабораторну

```
def find(lr:str)->Лабораторна
```

— у *Студента* додаємо метод, що додає лабораторну

```
def add(lr:str)->Лабораторна
```

— у *Лабораторну* додаємо метод, що додає спробу

```
def load(num:int,points:int)->int
```

Оскільки *Студент* зберігає кількість зданих лабораторних, то метод Лабораторної *load* повинен повертати 0 або 1 – чи змінилась кількість зданих лабораторних.

Додавання в Лабораторну Спроби алгоритмом також не відрізняється:

знайти спробу за номером листа

якщо знайшли – помилка

додати спробу

оновити статистичні дані (результат лабораторної)

Звідси

— у *Лабораторну* додаємо метод, що шукає спробу

```
def find(num :int)->Спроба
```

— у *Лабораторну* додаємо метод, що додає спробу

```
def load(num :int)->int
```

цей метод має повертати 1, якщо після додавання спроби лабораторна зі стану «не здано» перейшла в стан «здано»

Варто підкреслити, що для числових полів ці методи приймають значення, приведені до числового типу.

Хто відповідає за перевірку коректності значень полів? Той клас, який ці дані безпосередньо зберігає.

В який момент відбувається перевірка? Під час конструювання.

Щодо пошуку. Якщо студент однозначно задається номером заліковки, то реалізувати пошук не так складно: циклом пройти контейнер. Але це не найкращий спосіб. Набагато краще для класів означити оператор == (він має порівнювати ті поля, що однозначно характеризують об'єкт, наприклад заліковки студентів, коди для лабораторних, тощо) і використовувати вбудовані в контейнер методи пошуку.

Тоді пошук студента буде виглядати приблизно так

```
контейнер_студентів.index(Студент(номер_заліковки))
```

Що варто протестувати: усі додані методи.

Під час тестування можна до методів, що виконуються завантаження, додати виведення повідомлень типу:

«я метод такий-то класу такого-то працюю над аргументами такими-то»

Що ще має сенс додати до класів: метод *__repr__*, щоб мати можливість побачити дані не тільки під налагоджувачем.

6 Диспетчер завантаження даних

Ця частина стосується тільки підходу №1 виконання лабораторної (використання ООП), див. частину *4 Шляхи виконання лабораторної роботи* умови лабораторної.

Зміст етапу. Розбір вхідного файлу та завантаження його вмісту в об'єкт класу *Інформація*.

Диспетчер завантаження даних повинен отримувати файл та об'єкт класу *Інформація* та завантажувати дані з файлу в об'єкт.

Диспетчер має не тільки читати рядки файлу, але ще аналізувати порожні та «бракувати» ті, що мають невідповідну кількість полів, тощо. Отже, варто зробити його класом. Назвемо цей клас **Builder**.

Клас **Builder** можна зробити з конструктором без аргументів, а файл та об'єкт передавати як аргументи методу **load**. У такому випадку метод **load** повинен починати свою роботу з скидання поточного стану диспетчера.

У якому вигляді **Builder** приймає файл? Він може приймати вже відкритий для читання файл. Він може приймати шлях до файлу та кодування.

Як він виконує читання? На початку завантаження він створює ітератор, що повертає розібрані на поля* рядки файлу (доречно буде використати **csv.reader**).

*список полів, утворений за рядком, якщо використовувати **csv.reader**

Примітка. Має сенс «робочі» змінні зберігати в об'єкті, а не передавати між методами. Отже результат читання рядка має зберігатися в якійсь змінній об'єкта, скажімо **line**.

Завантаження даних виглядає як цикл по всіх рядках, у середині якого щось з отриманим рядком відбувається.

Після отримання чергового рядка у вигляді списку його полів слід:

— перевірити, чи порожній: порожній пропустити

— перевірити кількість полів

— записати поля з списку в окремі змінні об'єкта, одночасно з цим перетворюючи їх тип:

номер залікової книжки (рядок з 6 цифр) – без змін, бо рядок

навчальна група (рядок, не більше 6 символів) – без змін, бо рядок

порядковий номер листа з лабораторною роботою (додатне ціле) – перетворити на ціле

код лабораторної роботи (рядок, від 1 до 7 символів), – без змін, бо рядок

кількість балів за спробу (ціле від 0 до 10) – перетворити на ціле

— записати дані в об'єкт інформації, викликавши метод **load** класу *Інформація*.

Рядкові поля можна з списку й не переписувати, але викликати метод **load(line[0], line[1]…)** не дуже зручно: виникає бажання розуміти відповідність між полями та назвами. З цифровими позначками заплутатись легше.

Під час перетворення на ціле контролюється тільки те, щоб рядок був зображенням цілого. Діапазон отриманого цілого значення не контролюється.

Усі ці дії утворюють суть допоміжного методу *_обробити_поточний_рядок()*.

Дії, пов'язані з записом полів з списку в окремі змінні об'єкта, складають допоміжний метод *_конвертувати_поля()*.

Функція **load_data** виконує завантаження даних за допомогою об'єкта класу **Builder**.

7 Виведення даних: метод output

Метод **output** класу *Інформація* зараз приймає аргументи:
— ім’я вихідного файлу;
— кодування, в якому слід записати вихідний файл.

До класу *Інформація* має сенс додати прихований метод **_output**, що отримує підготовлений до запису потік.

У більшості варіантів, щоб знайти запитувані дані, їх має сенс відсортувати (in place (на місці), тобто не створюючи копій; метод **sort** сортує «на місці», функція **sorted** утворює копію).

Примітка. Для задач типу «знайти 5 найбільших» існують алгоритми, що вміють шукати 5-те за порядком значення у невідсортованій послідовності за лінійний час. Але використання таких бібліотек в лабораторній не планується.

Виведення теж має здійснюватися в певному порядку.

Скопіюємо умову, щоб вона була перед очима ☺

Знайти всіх студентів, що отримали щонайменше 8 балів за кожну лабораторну, що виконували. Вивести по кожному з них інформацію:
на першому рядку:
номер залікової книжки, група, загальна кількість листів з лабораторними
на наступних рядках, починаючи з табуляції, вивести по кожній зданій студентом лабораторній (по одній на рядок):
код лабораторної, номер листа з останньою найкращою спробою, кількість балів за спробу
у сортуванні: кількість балів за спробу (за спаданням), код лабораторної.

Гарантовано правильний за форматом[3] приклад вмісту вихідного файлу:

```
1000001     K00   3
      LR2   999   10
      LR1   546   9
      LR3   647   8
1000005     K00   4
      LR2   500   10
      LR3   221   9
      LR1   300   8
      LR4   777   8
```

Нагадую, що роздільниками мають бути табуляції.

"Перші" рядки відмічені жовтим. За ними слідують відповідні їм "наступні" рядки. Сортування стосується "наступних" рядків. Сортування "перших" рядків може бути довільним.

8 Порядок розробки

Загальну структуру програми, обробку командного рядка та основну функцію можна розробляти та тестувати окремо від інших частин коду.

Систему класів також можна розробляти паралельно. Тестування розпочати з класів нижнього рівня. З метою наладки може виявитись доречним у класах мати метод __repr__.

Диспетчер завантаження даних також може розроблятись паралельно. Спочатку зробити його таким, що імітує завантаження даних в об'єкт (тобто імітує виконання load або сам метод load класу Інформація зробити заглушкою). Мета? Перевірити правильність розбору вхідного файлу.

Далі це все можна збирати разом.

Наприкінці слід реалізувати виведення потрібної інформації та додати його до програми.

Рекомендовано:

1) окремо протестувати загальну структуру програми (обробка командного рядка, кроки роботи);
2) окремо протестувати класи, що зберігають дані, та методи класів, що додають у них дані;
3) окремо протестувати диспетчер завантаження (щоб переконатись, що він не спотворює інформацію);
4) протестувати отримання вихідної інформації.

[3] Зміст точно не відповідає вхідним даним прикладу. Але формат точно відповідає. І мета прикладу показати саме формат.