УКРАЇНСЬКИЙ КАТОЛИЦЬКИЙ УНІВЕРСИТЕТ

ФАКУЛЬТЕТ ПРИКЛАДНИХ НАУК

Комп’ютерні науки та Системний аналіз

**Назва проекту**

**“Власний варіант бібліотеки itertools”**

*Автори:*

*Софія Книшоїд*

*Ярослав Корч*

*Андрій Білінський*

*Дмитро Гребенюк*

*Тумоян Кірілл*

20 грудня 2021



1. **Вступ.**

Нашою ціллю була реалізація власних версій функцій з бібліотеки itertools.

В процесі роботи ми покращили свої навички з побудови та оцінки алгоритмів, а також спробували використати знання з дискретної математики на практиці. Для успішного написання цього проєкту нам були особливо корисними наступні теми та поняття з дискретної математики:

— декартовий добуток, алгоритм його знаходження;

— комбінаторне розміщення, принцип пошуку розміщень

— частковий випадок перестановок

— алгоритм знаходження комбінацій без повторень

— пошук комбінацій з повтореннями

Перед початком роботи, наша команда розподілила завдання, щоб виконання проєкту було ефективнішим. Ось список підзавдань, які ми виконували:

* Тумоян Кірілл:
* Завдання номер 2 (функція count)
* Завдання номер 4 (функція product)
* Дмитро Гребенюк:
* Завдання номер 5 (функція permutations)
* Завдання номер 6 (функція combinations)
* Софія Книшоїд:
* Завдання номер 3 (функція repeat)
* Написання звіту, підбір та аналіз інформації
* Андрій Білінський:
* Завдання номер 1 (функція count)
* Оцінка оптимальності алгоритмів, часова оцінка функцій
* Ярослав Корч:
* Завдання номер 7 (функція combinations\_with\_replacement)
* Реалізація та оптимізація перевірок unittest; розробка перевіркового модуля

1. **Псевдокод.**

Загалом, було реалізовано 7 аналогів функцій бібліотеки itertools:

* count
* cycle
* repeat
* product
* permutations
* combinations
* combinations\_with\_replacement

1. count(start=0, step=1)

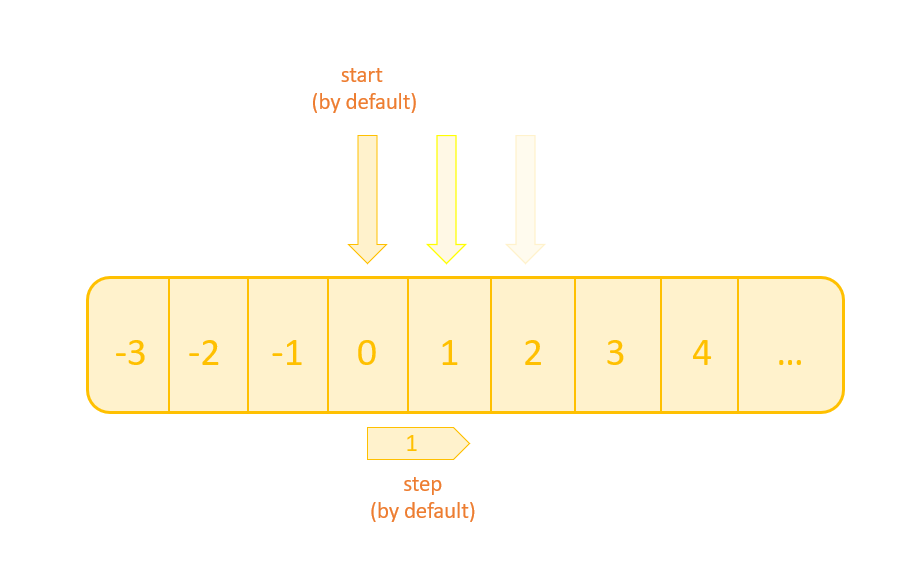
– функція повертає нескінченний ітератор цілих чисел;

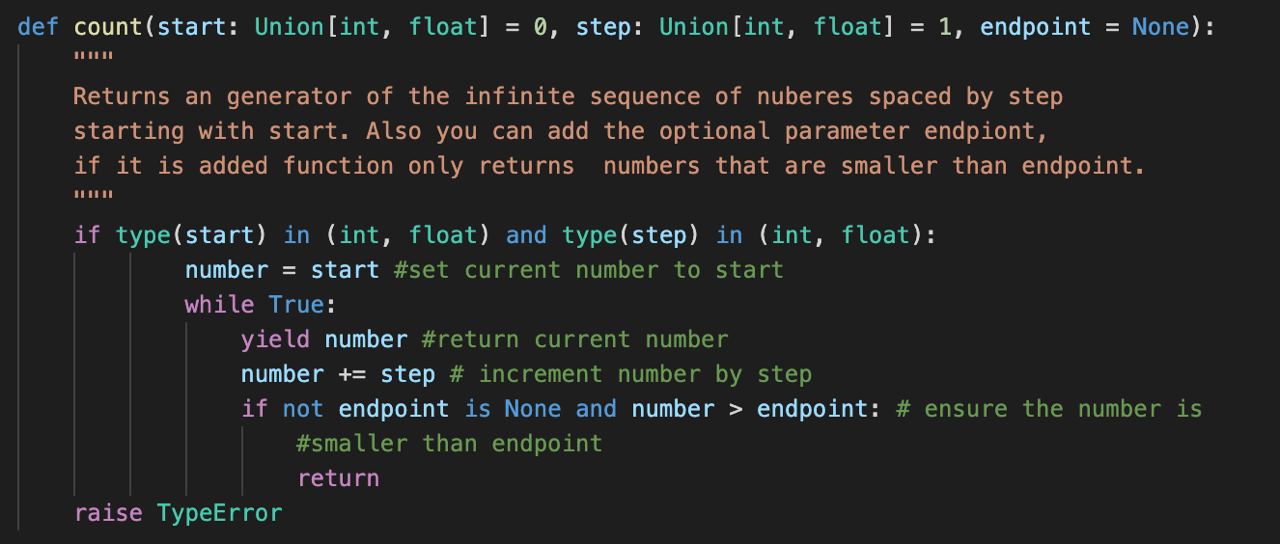
– аргумент start визначає початок рахунку, за замовчуванням отримує значення 0;

– аргумент step визначає, наскільки відрізнятиметься від попереднього кожне наступне згенероване число, за замовчуванням отримує значення 1;

– функція може викликатися без аргументів, при цьому буде повертати послідовність виду 0, 1, 2, 3, 4, …

– функція найчастіше використовується як частина більш складних алгоритмів, передається аргументом у функції zip() та map();





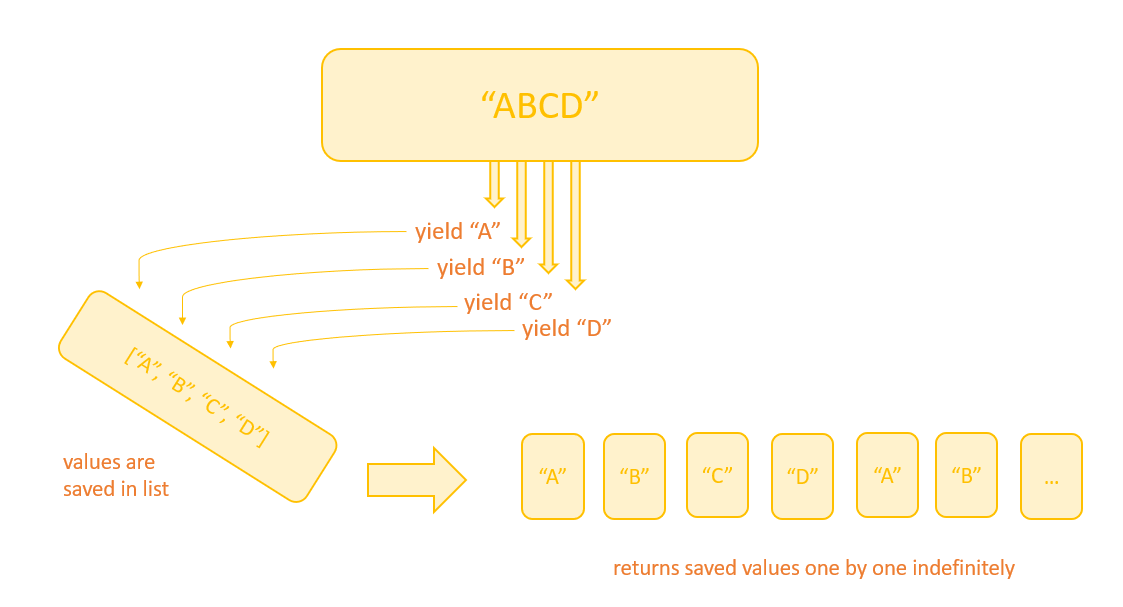
1. cycle (iterable)

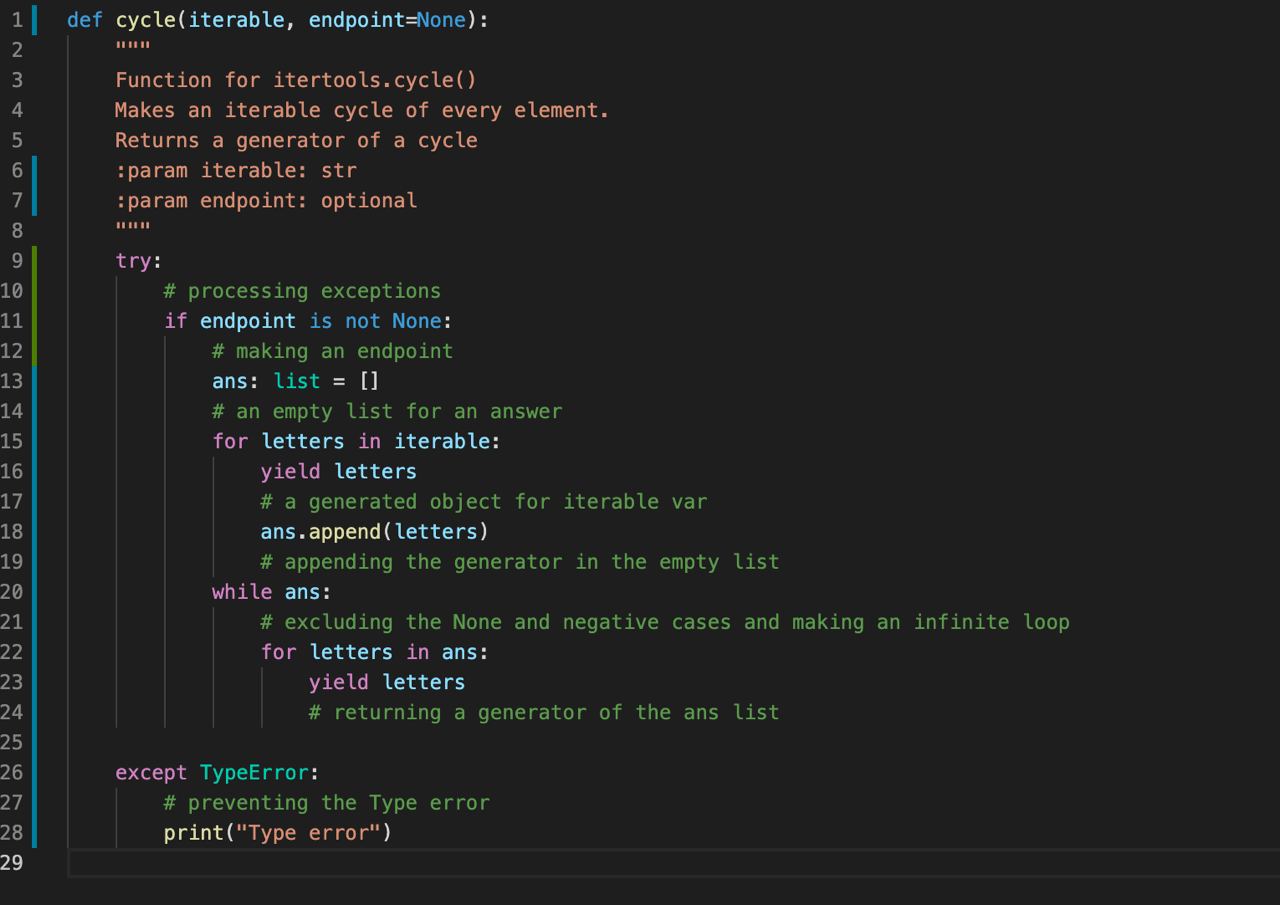
– повертає нескінченний ітератор з елементами переданого до функції ітератора;

– аргумент iterable представляє собою внутрішній ітератор, по якому буде проходити функція;

– функція не може викликатися без аргументу, обов’язково потрібно передати значення iterable;

– функція може використовувати значні об’єми пам’яті, на що слід зважати під час її використання у коді





1. repeat (value, stop\_point=None)

– повертає нескінченний ітератор значення value;

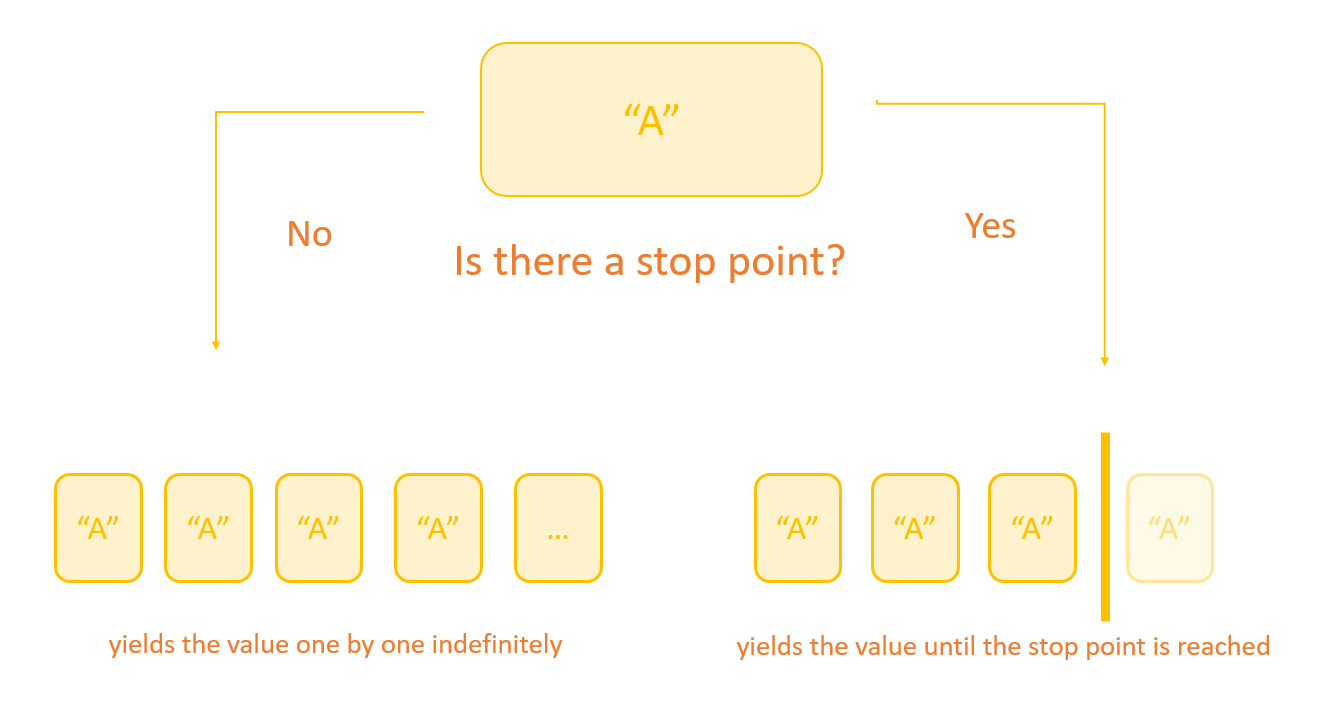
– аргумент value представляє значення, яке щоразу повертатиме функція;

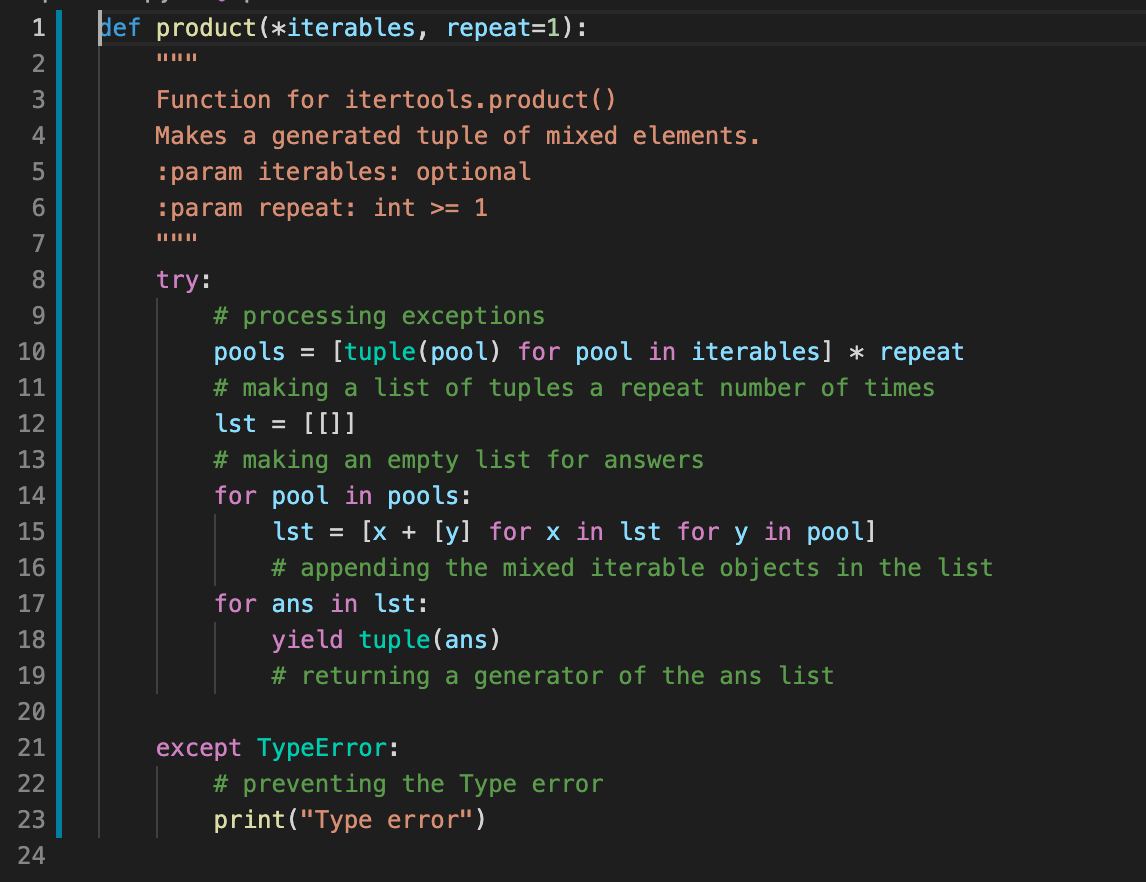
– аргумент stop\_point визначає останню ітерацію функції, з допомогою аргумента користувач регулює кількість разів повертання value;

– аргумент value є обов’язковим для роботи функції;

– аргумент stop\_point не є обов’язковим: за замовчуванням набуває значення None, і функція повертає value нескінченну кількість разів

– функція часто використовується як аргумент у функціях map() та zip()





1. product (\*iterables, repeat=1)

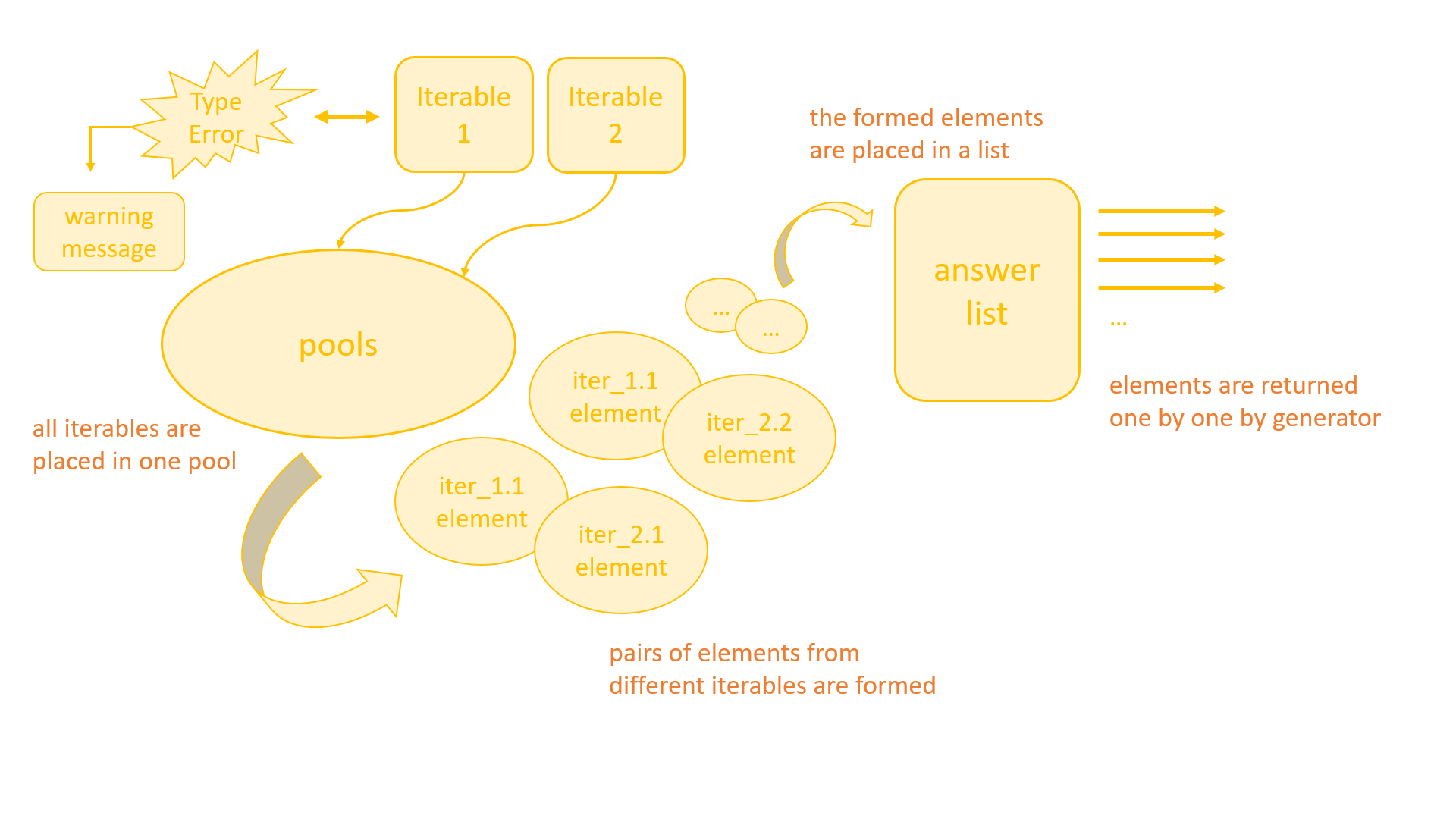
– повертає генератор декартового добутку аргументів \*iterables;

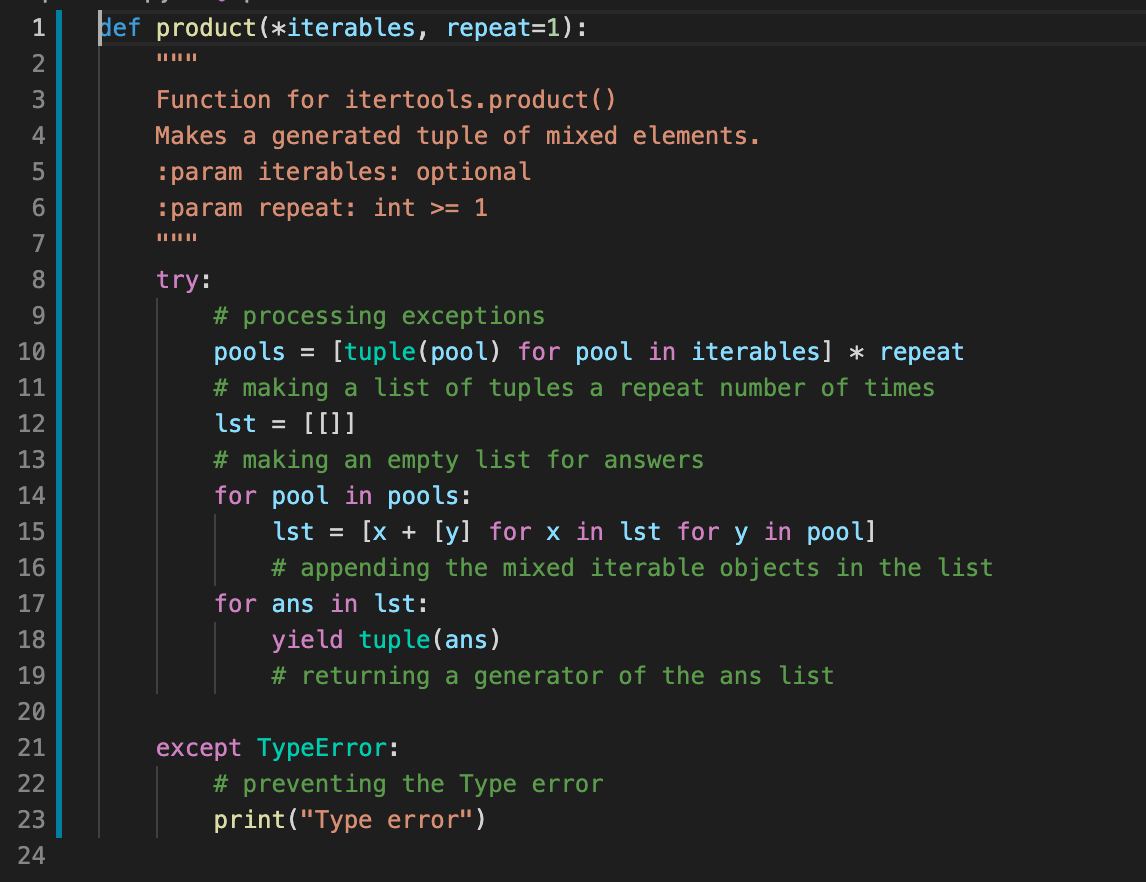
– функція працює аналогічно принципу вкладених for-циклів (тобто фактично бере перший елемент із першого ітерованого об’єкта, перебирає всі з другого, міняє перший, знову перебирає…);

– аргумент repeat означає кількість повторів того самого ітерованого об’єкта; передається функції необов’язково, за замовчуванням його значення = 1;

– реалізація функції запобігає виникненню небажаної помилки TypeError та повідомляє користувача, якщо дана проблема загрожувала роботі функції;

– функція корисна не лише при використанні більш складного коду: результат її роботи дозволяє користувачу розраховувати декартовий добуток, реалізувати модель лічильника, зручно опрацьовувати об’єкти, що вимагають збереження лексикографічного порядку тощо





1. permutations (iterable, length=None)

– повертає генератор розміщень елементів з ітерованого об’єкта;

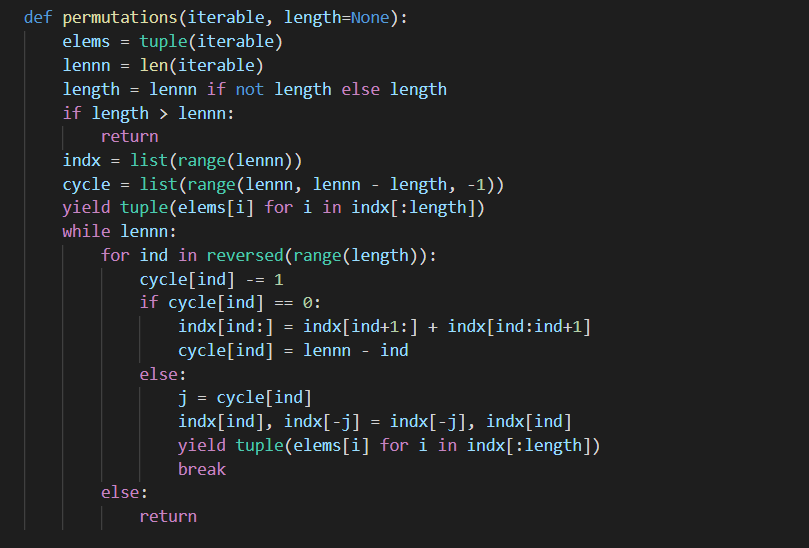
– аргумент iterable представляє ітерований об’єкт, з якого будуть братися розміщення;

– аргумент length вказує на довжину шуканих розміщень; не обов’язковий, за замовчуванням = None, тому у разі ігнорування цього аргумента шукаються перестановки елементів даного ітератора;

– функція значною мірою спрощує роботу під час обробки великої кількості даних у задачах, що зводяться до рамок застосування комбінаторики

– часова оцінка алгоритму:





1. combinations (iterable, r)

– повертає генератор комбінацій елементів з iterable по k;

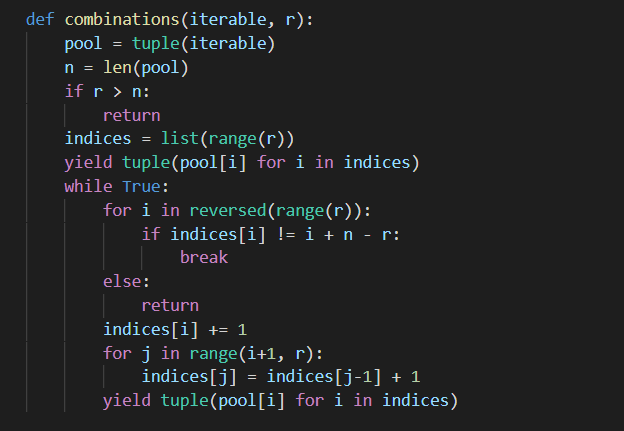
– аргумент r у функції вказує, по скільки елементів буде братися на комбінацію;

– аргумент iterable є ітерованим об’єктом, з якого будуть обиратися комбінації;

– комбінації утворені алгоритмом зберігають посортованість, тому дану функцію зручно використовувати, якщо відсортованість елементів є важливою;

– часова оцінка алгоритму:





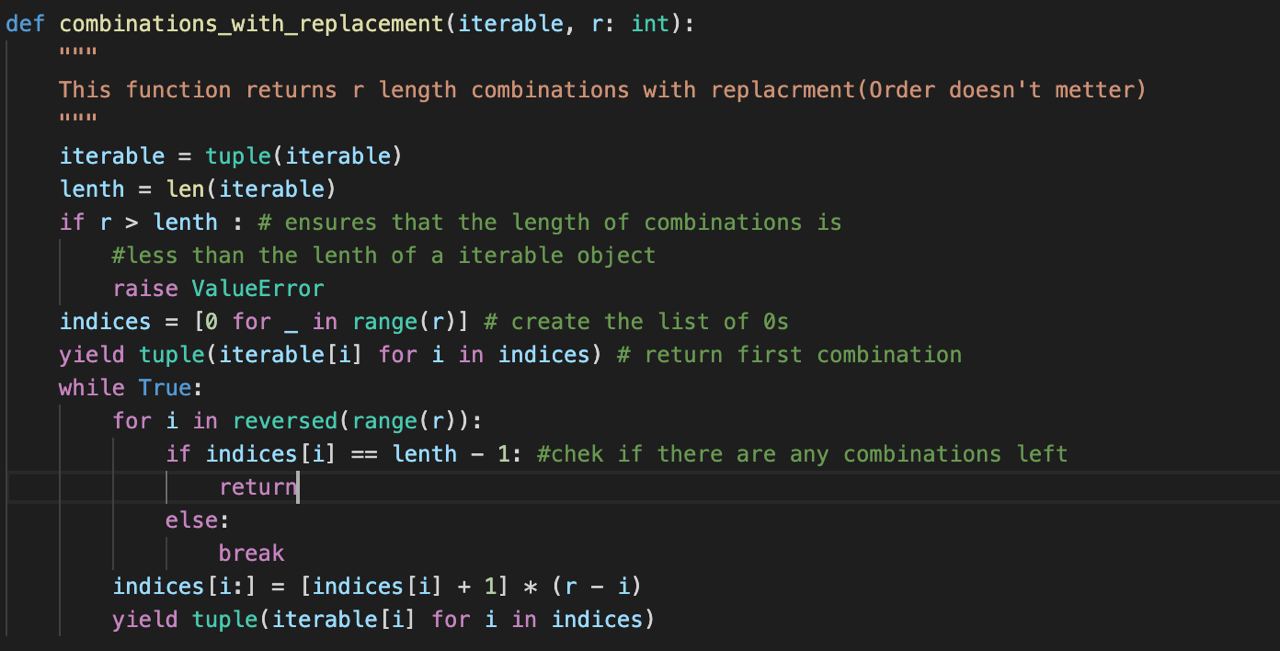
1. combinations\_with\_replacement(iterable, r)

– функція повертає генератор комбінацій з iterable по r, до того ж елементи можуть повторюватися;

– аргумент функції r надає алгоритму довжину підпослідовностей, комбінацій, які шукаються;

– аргумент функції iterable надає алгоритму ітерований об’єкт, з якого будуть обиратися комбінації;

– комбінації, отримані в результаті роботи цього алгоритму посортовані, якщо початковий аргумент iterable був також посортований



1. **Висновки.**

Якщо підсумувати результати нашої роботи над цим проєктом, можна сказати, що реалізація завдання була корисною.

Коли ми писали потрібні функції, нам дуже часто ставали у пригоді знання з дискретної математики. Можливо, використання аналогій із поняттями з цього курсу не завжди є очевидними, та вони значно спрощують роботу. Ймовірно, такі паралелі можна проводити у значно більшій кількості задач, ніж могло би здатися на перший погляд.

Виконання завдання великого обсягу показало, що тестування функцій є не менш важливою частиною роботи, ніж їх написання. Крім того, робота у групі людей зробила очевидною користь написання документації: якщо функція зрозуміло описана, значно легше розібратися у написаному коді та зрозуміти принцип дії функції.

Загалом, нам вдалося досягти поставлених цілей та реалізувати потрібні функції.