

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



APPLIED  
SCIENCES  
FACULTY ●

## 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 Книшюїд:
  - 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

## **2. Псевдокод.**

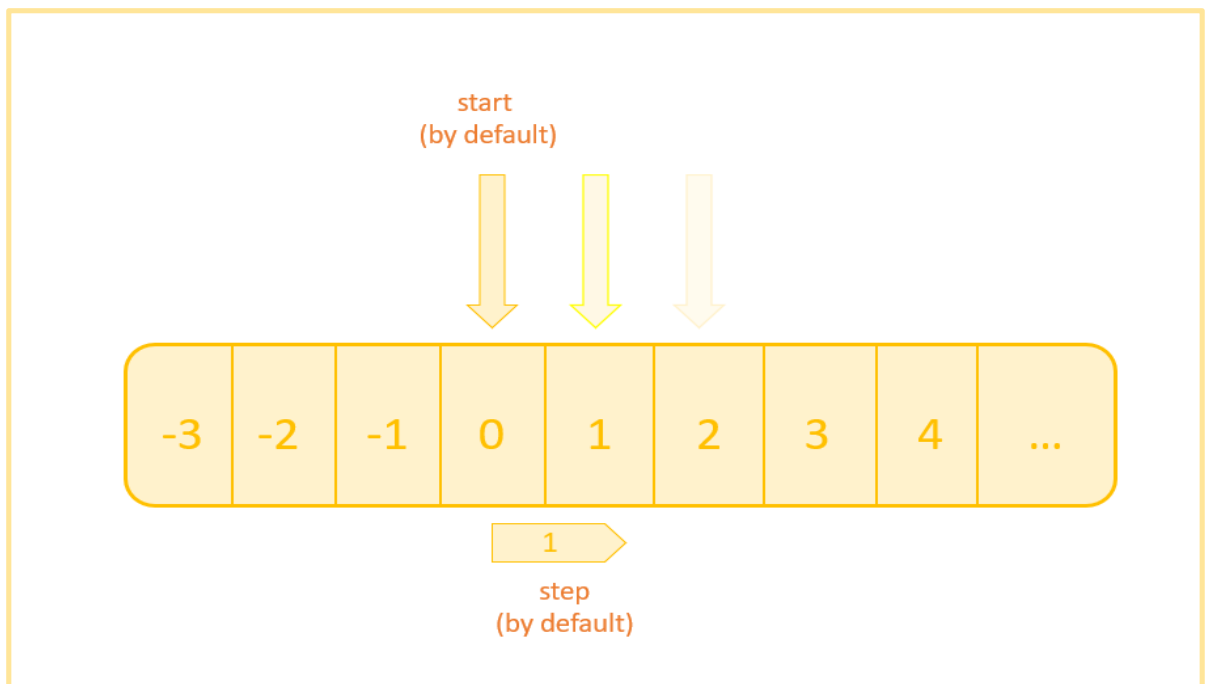
Загалом, було реалізовано 7 аналогів функцій бібліотеки itertools:

- 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;

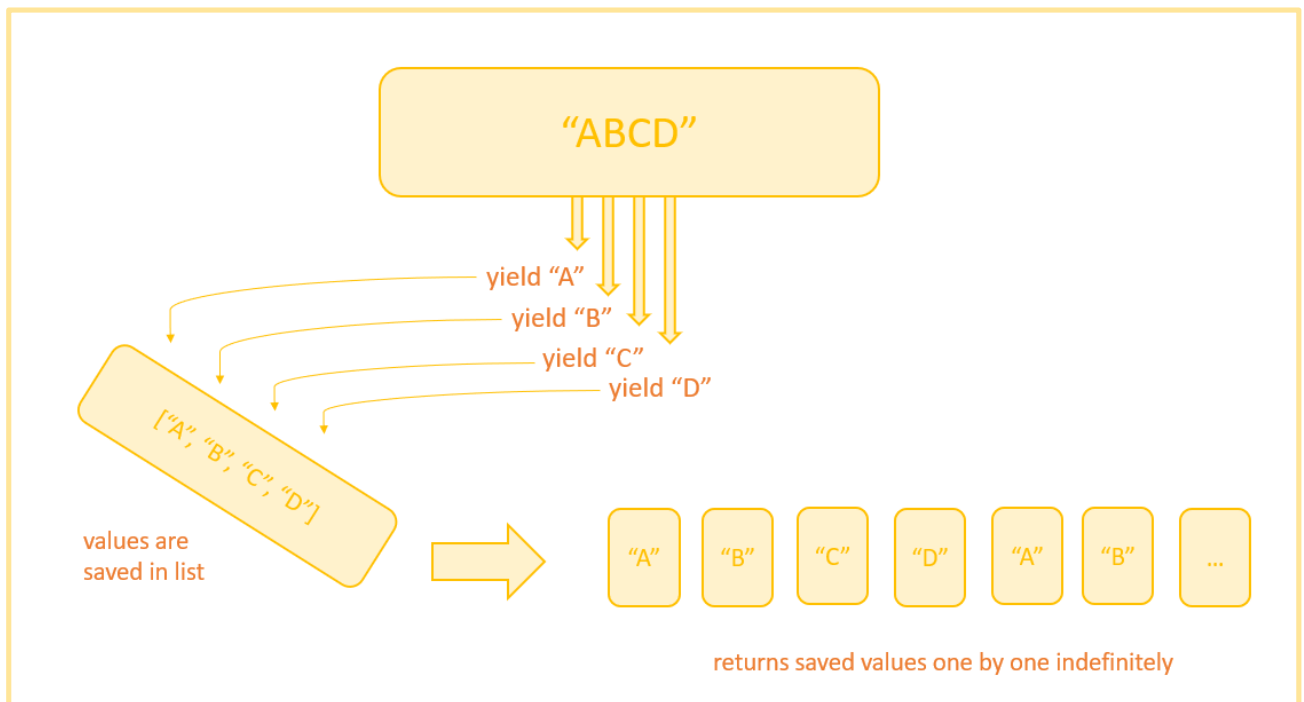
- аргумент `step` визначає, наскільки відрізнятиметься від попереднього кожне наступне згенероване число, за замовчуванням отримує значення 1;
- функція може викликатися без аргументів, при цьому буде повертати послідовність виду 0, 1, 2, 3, 4, ...
- функція найчастіше використовується як частина більш складних алгоритмів, передається аргументом у функції `zip()` та `map()`;



```
def count(start: Union[int, float] = 0, step: Union[int, float] = 1, endpoint = None):  
    """  
    Returns an generator of the infinite sequence of nuberess spaced by step  
    starting with start. Also you can add the optional parameter endpoint,  
    if it is added function only returns numbers that are smaller than endpoint.  
    """  
    if type(start) in (int, float) and type(step) in (int, float):  
        number = start #set current number to start  
        while True:  
            yield number #return current number  
            number += step # increment number by step  
            if not endpoint is None and number > endpoint: # ensure the number is  
                #smaller than endpoint  
                return  
    raise TypeError
```

## 2. 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



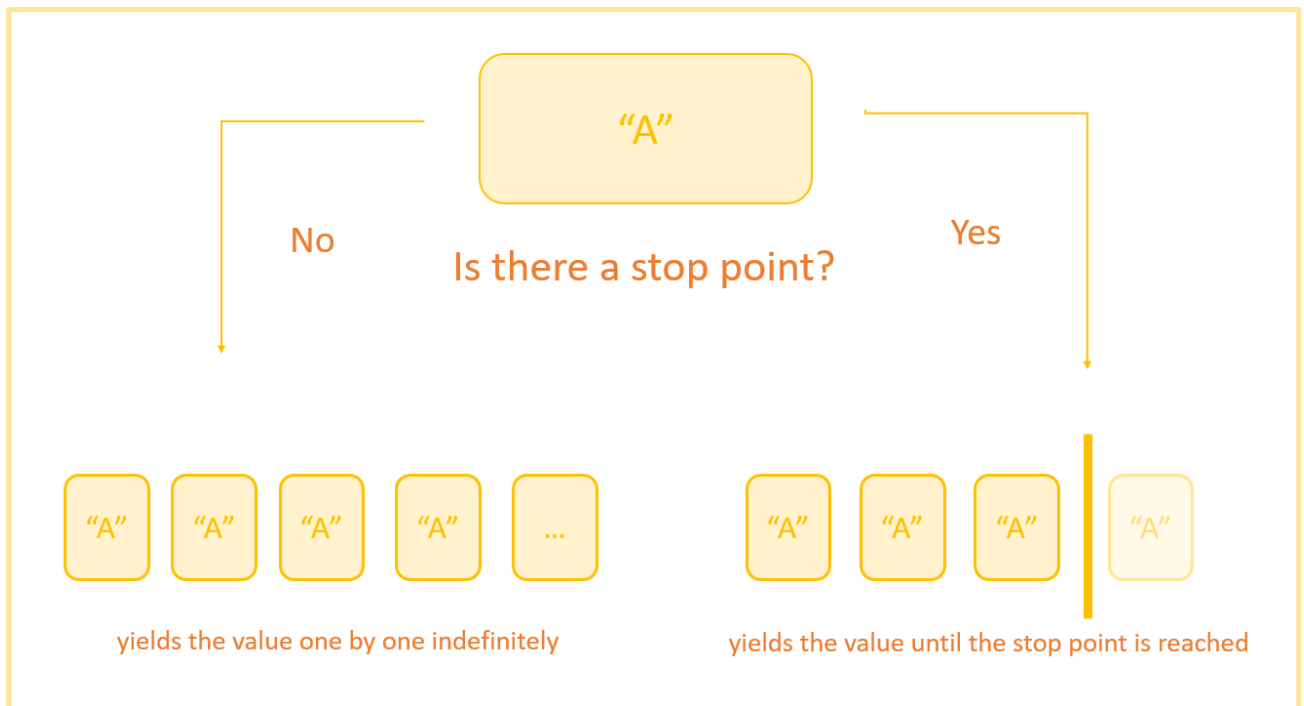
```

1  def cycle(iterable, endpoint=None):
2      """
3      Function for itertools.cycle()
4      Makes an iterable cycle of every element.
5      Returns a generator of a cycle
6      :param iterable: str
7      :param endpoint: optional
8      """
9      try:
10         # processing exceptions
11         if endpoint is not None:
12             # making an endpoint
13             ans: list = []
14             # an empty list for an answer
15             for letters in iterable:
16                 yield letters
17                 # a generated object for iterable var
18                 ans.append(letters)
19                 # appending the generator in the empty list
20             while ans:
21                 # excluding the None and negative cases and making an infinite loop
22                 for letters in ans:
23                     yield letters
24                 # returning a generator of the ans list
25
26     except TypeError:
27         # preventing the Type error
28         print("Type error")
29

```

### 3. 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()



```
def repeat(value, stop_point=None):  
    """  
    The following function mirrors the behaviour of eponymous  
    itertools repeat function.  
    It takes two arguments:  
    --- "value" argument (a required one)  
    | takes a value that is to be repeated by function  
    --- "stop_point" argument (an optional one)  
    | defines a number at which repeating should stop;  
    | if not passed: function repeats the value infinitely;  
    | by default is None  
    Function either repeats the value infinitely,  
    or repeats the value up until reaching the stop point.  
    The function can save time and effort  
    in writing long complicated tasks.  
    The function is useful for "zip" and "map" usage.  
    >>> for _ in repeat(8,3): print (_)  
    8  
    8  
    8  
    """"  
    if stop_point is not None:  
        for _ in range(stop_point):  
            yield value  
    else:  
        while True:  
            yield value
```

4. `product(*iterables, repeat=1)`

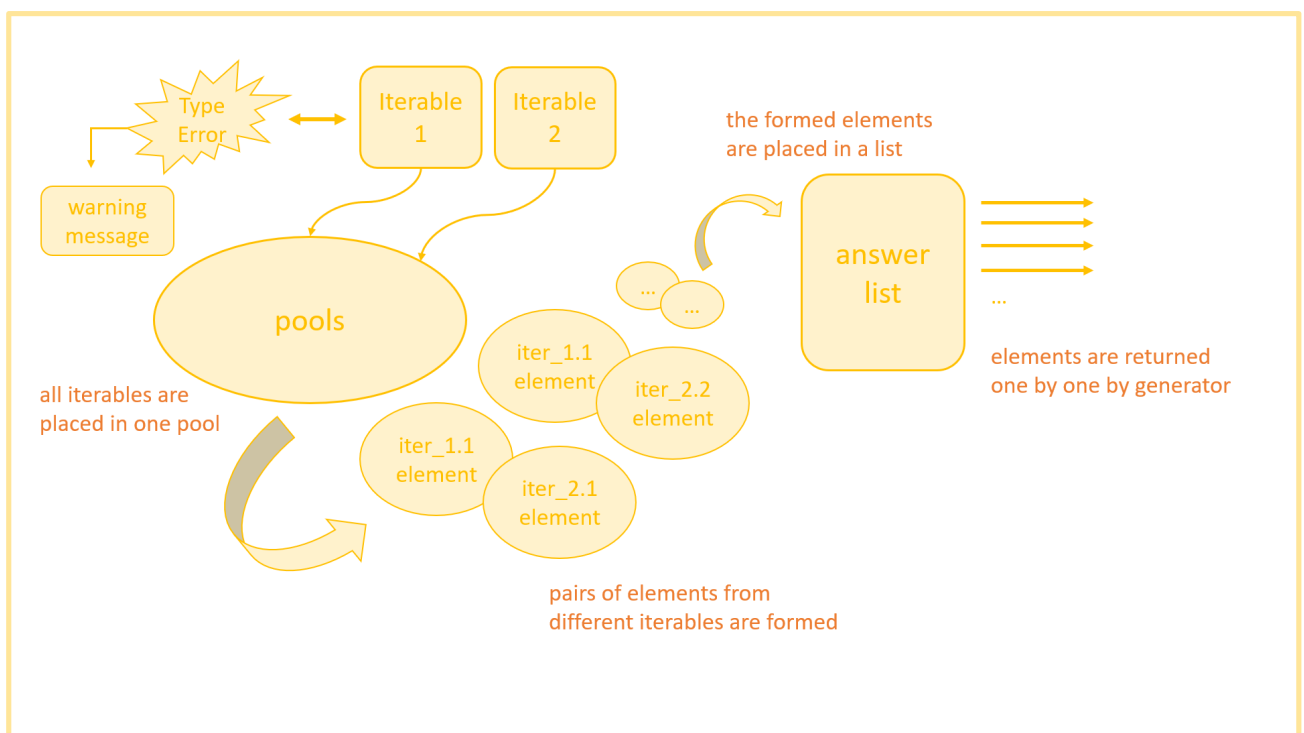
– повертає генератор декартового добутку аргументів \*iterables;

– 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  def product(*iterables, repeat=1):
2      """
3      Function for itertools.product()
4      Makes a generated tuple of mixed elements.
5      :param iterables: optional
6      :param repeat: int >= 1
7      """
8      try:
9          # processing exceptions
10         pools = [tuple(pool) for pool in iterables] * repeat
11         # making a list of tuples a repeat number of times
12         lst = [[]]
13         # making an empty list for answers
14         for pool in pools:
15             lst = [x + [y] for x in lst for y in pool]
16             # appending the mixed iterable objects in the list
17         for ans in lst:
18             yield tuple(ans)
19             # returning a generator of the ans list
20
21     except TypeError:
22         # preventing the Type error
23         print("Type error")
24

```

## 5. 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:

```

original algorithm of permutations:      2.6226043701171875e-06
our version:                            2.86102294921875e-06

```



```

def combinations(iterable, r):
    pool = tuple(iterable)
    n = len(pool)
    if r > n:
        return
    indices = list(range(r))
    yield tuple(pool[i] for i in indices)
    while True:
        for i in reversed(range(r)):
            if indices[i] != i + n - r:
                break
        else:
            return
        indices[i] += 1
        for j in range(i+1, r):
            indices[j] = indices[j-1] + 1
        yield tuple(pool[i] for i in indices)

```

## 7. 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 підпоследовностей, 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

```

def combinations_with_replacement(iterable, r: int):
    """
    This function returns r length combinations with replacrmnt(Order doesn't metter)
    """
    iterable = tuple(iterable)
    lenth = len(iterable)
    if r > lenth : # ensures that the length of combinations is
        #less than the lenth of a iterable object
        raise ValueError
    indices = [0 for _ in range(r)] # create the list of 0s
    yield tuple(iterable[i] for i in indices) # return first combination
    while True:
        for i in reversed(range(r)):
            if indices[i] == lenth - 1: #chek if there are any combinations left
                return
            else:
                break
        indices[i:] = [indices[i] + 1] * (r - i)
        yield tuple(iterable[i] for i in indices)

```

### 3. Висновки.

Якщо підсумувати результати нашої роботи над цим проектом, 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.

