Zaawansowane techniki programowania w języku Python

Lista zadań – Iteratory

Zadanie 1 - Count

Zaimplementuj własny iterator o nazwie `count`, zwracający kolejne liczby całkowite. Liczba, od której należy zacząć odliczanie, powinna być podana jako argument dla funkcji `__init__`. Iterator ten jest nieskończony (infinite iterator), tzn. funkcja `__next__` zawsze zwróci kolejny element.

Napisz program, w którym wykorzystasz swój iterator do wypisania na ekranie kolejnych 10 liczb całkowitych. Niech liczbę, od której należy zacząć odliczanie, podaje użytkownik.

Zadanie 2 – Tetranacci

Zaimplementuj własny iterator o nazwie `tetranacci`[1], zwracający kolejne liczby ciągu Tetranacciego. Funkcja `__init__` powinna posiadać parametr o nazwie `steps` określający liczbę wyrazów ciągu, po których funkcja `__next__` rzuca wyjątek `StopIteration`.

Napisz program, w którym wykorzystasz swój iterator do wypisania na ekranie kolejnych wyrazów tego ciągu. Niech liczbę wyrazów do wypisania podaje użytkownik.

Zadanie 3 – Repeat

Zaimplementuj własny iterator o nazwie `repeat`, zwracający obiekt podany przez użytkownika (parametr funkcji `__init__` o nazwie `elem`) dokładnie `times` razy. Jeśli wartość parametru `times` nie została określona, iterator powinien zwracać wartości w nieskończoność.

Przykład:

1 https://pl.wikipedia.org/wiki/Ci%C4%85g_Fibonacciego#Ci.C4.85g_.E2.80.9ETetranacciego.E2.80.9D

Notatki:

```
repeat(10, 3) → 10 10 10
repeat(10, 5) → 10 10 10 10 10
repeat(5) → 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5…
repeat(5, None) → 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5…
```

Zadanie 4 – Odd first

Zaimplementuj własny iterator o nazwie `odd_first`, zwracający elementy listy znajdujące się na pozycjach nieparzystych, a następnie na parzystych.

Przykład:

```
odd_first(["A", "B", "C", "D", "E"]) → A C E B D
```

Zadanie 5 - Chain

Zaimplementuj własny iterator o nazwie `chain`, zwracający kolejne elementy z podanych jako argument obiektów iterowalnych.

Przykład:

```
chain('ABC', 'DEF') → A B C D E F
chain([1, 2, 3], [4, 5, 6], [7, 8, 9]) → 1 2 3 4 5 6 7 8 9
chain("ABC", [1, 2, 3], [], "DEF") → A B C 1 2 3 D E F
```

Możesz wykorzystać (między innymi):

- *args,
- funkcję iter(),
- mechanizm obsługi wyjątków try-except.

Iteratory **count**, **repeat**, **chain** są częścią modułu **itertools**[2].

2 zobacz: itertools — Functions creating iterators for efficient looping, https://docs.python.org/3/library/itertools.html

Notatki:

.....

.....