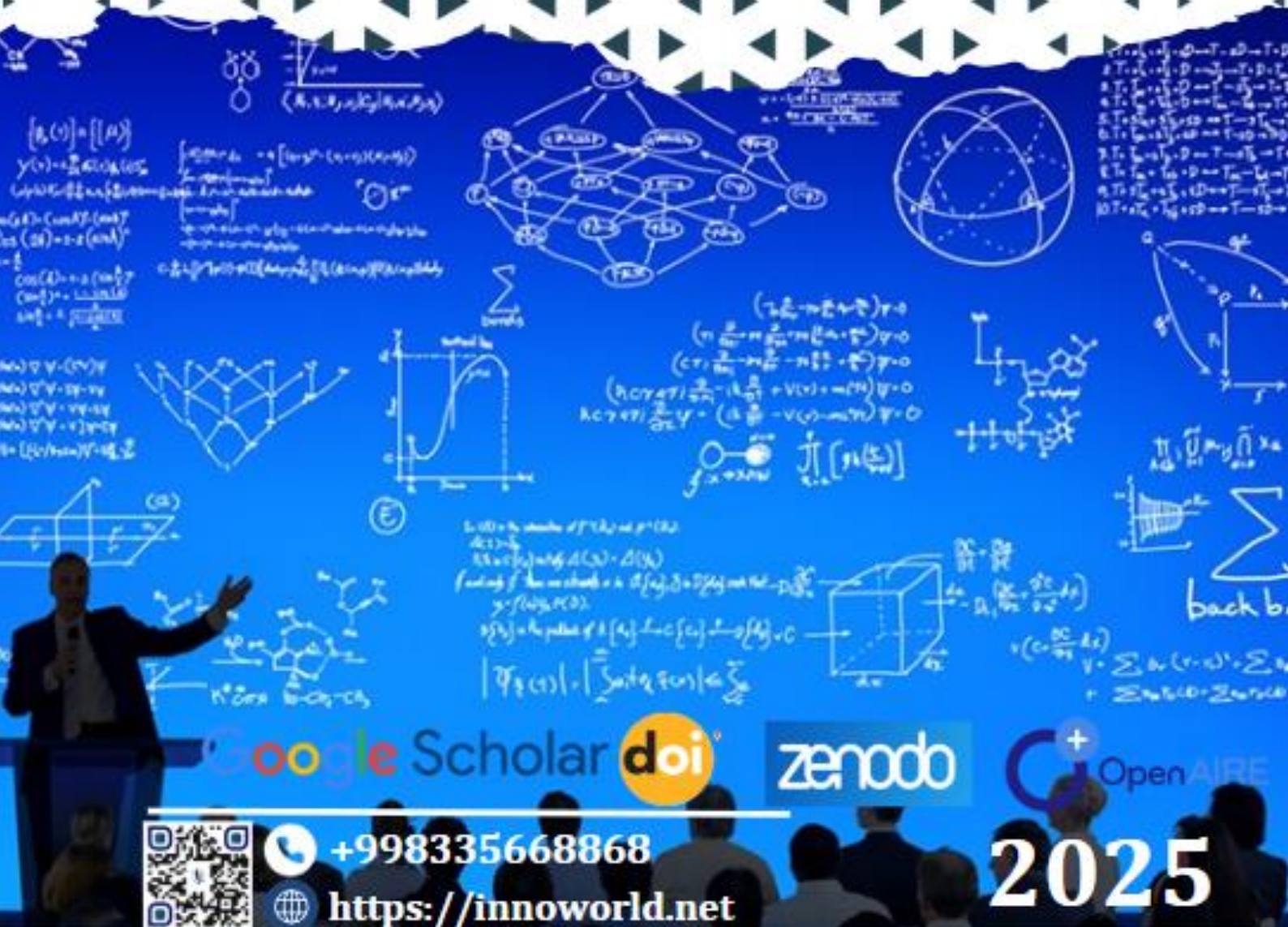
INNOVATIVE WORLD
Ilmiy tadqiqotlar markazi

# ZAMONAVIY ILM-FAN VA TA'LIM: MUAMMO VA YECHIMLAR

## ILMIY-AMALIY KONFERENSIYA



Google Scholar doi zenodo OpenAIRE

+998335668868
https://innoworld.net

2025

**«INNOVATIVE WORLD» ILMIY TADQIQOTLARNI QO'LLAB-QUVVATLASH MARKAZI**

# «ZAMONAVIY ILM-FAN VA TADQIQOTLAR: MUAMMO VA YECHIMLAR» NOMLI 2025-YIL № 4-SONLI ILMIY, MASOFAVIY, ONLAYN KONFERENSIYASI

**ILMIY-ONLAYN KONFERENSIYA TO'PLAMI**
**СБОРНИК НАУЧНЫХ-ОНЛАЙН КОНФЕРЕНЦИЙ**
**SCIENTIFIC-ONLINE CONFERENCE COLLECTION**

**www.innoworld.net**

**O'ZBEKISTON-2025**

# Iterator va generatorlarni amaliyotda yozish

**Yusupov Mirsaid**
Farg'ona davlat universiteti Amaliy matematika
va informatika kafedrasi o'qituvchisi
E-mail: mirsaidbeky@gmail.com
ORCID- 0009-0001-1989-0088
**Sobirjonov Siyavushxo'ja Suxrobxo'ja o'g'li**
Farg'ona davlat universiteti Amaliy matematika
yo'nalishi 3-bosqich 23.07-guruh talabasi
E-mail: siyavushxujayev@gmail.com

**Annotatsiya:** Ushbu maqolada Python dasturlash tilida iterator va generatorlarning ishlash prinsiplari, ularni amaliyotda qo'llash usullari hamda samaradorlik jihatlari yoritiladi. Keltirilgan misollar orqali __iter__() metodi yordamida maxsus iterator obyektlarini yaratish va yield operatori asosida generator funksiyalarni ishlab chiqish jarayoni bosqichma-bosqich tushuntiriladi. Iterator va generatorlarning xotira tejamkorligi, kechiktirilgan hisoblash (lazy evaluation) imkoniyatlari hamda katta hajmdagi ma'lumotlar bilan ishlashdagi afzalliklari amaliy misollar bilan asoslab beriladi. Maqola Python'da oqimli ma'lumotlarni qayta ishlash, sikllar bilan ishlash va samarali kod yozish bo'yicha boshlang'ich hamda amaliy ko'nikmalarni shakllantirishga qaratilgan.

**Kalit so'zlar:** iterator, generator, kechiktirilgan hisoblash, lazy evaluation, xotira tejamkorligi, Python sikllari, ma'lumot oqimi, iterable obyekt, funksiya generatori, amaliy dasturlash.

**Annotation:** This article explores the principles and practical implementation of iterators and generators in the Python programming language. It explains how custom iterator objects can be created using the __iter__() and __next__() methods, as well as how generator functions operate through the yield keyword. The study highlights the memory efficiency and lazy evaluation features of generators, demonstrating their advantages when working with large data streams. Practical code examples are provided to illustrate how iterators and generators improve performance and simplify data processing in real-world programming scenarios.

**Keywords:** iterator, generator, __iter__(), __next__(), yield, lazy evaluation, memory efficiency, data streams, Python programming, iterable objects, generator functions.

**Аннотация:** В данной статье рассматриваются принципы работы итераторов и генераторов в языке программирования Python, а также их практическое применение. Подробно объясняется создание пользовательских итераторов с использованием методов __iter__() и __next__(), а также функционирование генераторных функций на основе ключевого слова yield. Особое внимание уделяется таким

преимуществам генераторов, как ленивые вычисления и экономия памяти, что делает их эффективными при обработке больших объёмов данных. Приведённые примеры демонстрируют, как итераторы и генераторы упрощают работу с потоками данных и повышают производительность программ.

**Ключевые:** итератор, генератор, __iter__(), __next__(), yield, ленивые вычисления, экономия памяти, поток данных, итерируемый объект, генераторная функция, Python.

**Kirish**

Python dasturlash tilida ma'lumotlar ustida ketma-ket ishlash jarayonida iterator va generatorlardan foydalanish muhim ahamiyatga ega. Iteratorlar yordamida obyektlarni bosqichma-bosqich o'qish, ularning har bir elementini tartib bilan qaytarish hamda sikllar orqali boshqarish mumkin bo'ladi. Generatorlar esa yield operatori asosida ishlaydigan, xotira tejamkorligi va kechiktirilgan hisoblash (lazy evaluation) imkoniyatiga ega bo'lgan maxsus funksiyalar bo'lib, katta hajmdagi ma'lumotlar bilan ishlashda samaradorlikni ta'minlaydi. Ushbu mavzu Pythonning ichki mexanizmlarini chuqurroq anglash, samarali kod yozish hamda ma'lumot oqimlari bilan ishlash ko'nikmalarini shakllantirish uchun muhim hisoblanadi. Mazkur maqolada iterator va generatorlarning asosiy tushunchalari, ularning amaliy qo'llanilishi va afzalliklari bosqichma-bosqich yoritib beriladi.

**Iterator va generatorlar: asosiy tushuncha**

Python dasturlash tilida iterator va generatorlar ma'lumotlar ustida ketma-ketlik asosida ishlashni soddalashtiradigan ichki mexanizmlardir. Iterator — bu har safar navbatdagi elementni qaytarib beradigan obyekt bo'lib, u ikki asosiy metodga ega: __iter__() va __next__(). __iter__() metodi obyektning iteratsiya uchun tayyor ekanini bildirsa, __next__() navbatdagi qiymatni qaytaradi va har bir chaqirilganda ichki ko'rsatkich bir bosqich oldinga siljiydi. Elementlar tugaganda StopIteration istisnosi ko'tariladi. Iteratorlar ma'lumotlarni tartib bilan qayta ishlash, sikllarni moslashuvchan boshqarish va katta massivlar bilan ishlashda qulaylik yaratadi.

Generatorlar esa iteratorlarning qulay alternativi bo'lib, ular yield operatori orqali bir vaqtning o'zida bitta qiymat qaytaradi va funksiyaning bajarilishini vaqtincha to'xtatadi. yield qayta chaqirilganda esa funksiya avval to'xtagan joyidan davom etadi. Bu jarayon kechiktirilgan hisoblash (lazy evaluation) deb ataladi. Generatorlarning asosiy afzalligi — xotira tejamkorligi, chunki ular butun ro'yxatni birdaniga yaratmaydi, balki elementlarni navbat bilan ishlab chiqaradi.

Iterator va generatorlardan foydalanish katta hajmdagi ma'lumotlarni qayta ishlash, fayllarni qadam-baqadam o'qish, oqimli ma'lumotlarni boshqarish va samarali kod yozishda juda foydalidir. Amaliyotda ular sikllar, ma'lumotlar oqimi, filtrlash, hisoblash jarayonlarida keng qo'llanadi va dastur samaradorligini sezilarli darajada oshiradi.

**Iteratorlarning vazifalari**

1. **Ketma-ket elementlarni olish** – iteratorlar obyektning har bir elementini navbat bilan qaytaradi.
2. **Elementlarni boshqarish** – sikllar orqali ma'lumotlarni tartib bilan qayta ishlash imkonini beradi.
3. **Ma'lumotlarni takrorlash (traversal)** – ro'yxat, lug'at, to'plam kabi iterable obyektlarni oson aylanadi.
4. **Katta hajmdagi ma'lumotlarni boshqarish** – barcha elementlarni bir vaqtning o'zida xotirada saqlamay, ketma-ket ishlash imkonini beradi.
5. **Moslashuvchanlik** – o'z iteratsion obyektlaringni yaratish va maxsus iteratsiya qoidalarini belgilash.

**Generatorlarning vazifalari**

1. **Kechiktirilgan hisoblash (lazy evaluation)** – qiymatlarni faqat kerak bo'lgan paytda hisoblaydi, xotirani tejamlaydi.
2. **Soddalashtirilgan iterator yaratish** – __iter__() va __next__() metodlarini qo'lda yozmasdan generator orqali navbatma-navbat qiymatlarni qaytarish.
3. **Ma'lumot oqimlarini boshqarish** – fayllar, sensorlar yoki oqimdagi katta ma'lumotlarni ketma-ket ishlash.
4. **Murakkab hisoblashlarni modulga ajratish** – har bir yield orqali qadam-baqadam natijalarni qaytarish, dastur samaradorligini oshirish.
5. **Sikl va hisoblashlarni optimallashtirish** – iteratsiya jarayonini qisqa va tushunarli kod orqali amalga oshirish.

**Kod misollar**

**1.** Iterator yaratish (**__iter__** va **__next__**)

```
class MyIterator:
    def __init__(self, data):
        self.data = data
        self.index = 0
    def __iter__(self):
        return self
    def __next__(self):
        if self.index < len(self.data):
            result = self.data[self.index]
            self.index += 1
            return result
        else:
            raise StopIteration
my_list = [1, 2, 3]
it = MyIterator(my_list)
for item in it:
    print(item)
```

**Natija:**

**2.** Generator yaratish (**yield**)

```
def my_generator(n):
    for i in range(n):
        yield i

for num in my_generator(3):
    print(num)
```

**Natija:**

0
1
2

**Taqqoslash: Iterator vs Generator**

| **Xususiyat** | **Iterator** | **Generator** |
|---|---|---|
| Yaratish usuli | __iter__() va __next__() metodlari | yield operatori bilan funksiya |
| Xotira ishlatish | Ba'zida ko'proq xotira talab qiladi | Lazy evaluation, xotira tejamkor |
| Kod soddaligi | Ko'proq kod talab qiladi | Kamroq va tushunarli kod |
| Qaytaradigan qiymat | __next__() chaqirilganda | yield chaqirilganda |
| Katta ma'lumotlar bilan ishlash | Murakkab bo'lishi mumkin | Juda qulay |

**Amaliy misol**

Tasavvur qilaylik, bizda katta hajmdagi raqamlar ro'yxati bor va faqat **juft sonlar**ni qaytarishimiz kerak:

```
def even_numbers(n):
    for i in range(n):
        if i % 2 == 0:
            yield i

for num in even_numbers(10):
    print(num)
```

**Natija:**

0
2
4
6
8

Bu misol xotira tejamkorligini ko'rsatadi, chunki generator faqat kerakli elementlarni qaytaradi, butun ro'yxatni xotirada saqlamaydi.

**Xulosa**

Python dasturlash tilida iterator va generatorlar **ketma-ketlik asosida ma'lumotlarni boshqarish va samarali ishlash** imkonini beradi.

- **Iteratorlar**: obyektning elementlarini navbat bilan qaytaradi, moslashuvchanlik va boshqaruv imkoniyatini beradi.
- **Generatorlar**: xotira tejamkor va kodni soddalashtiradi, kechiktirilgan hisoblashni amalga oshiradi.

Amaliyotda ular katta hajmdagi ma'lumotlar bilan ishlash, fayllarni oqim orqali o'qish va murakkab hisoblashlarni modulga ajratish uchun juda qulaydir. Shu bilan dastur samaradorligi va tushunarli kod yozish imkoniyati oshadi.

**FOYDALANILGAN ADABIYOTLAR**

1. Norman Matloff. *Tutorial on Python Iterators and Generators*. PDF. heather.cs.ucdavis.edu+1
2. Steven F. Lott. *Functional Python Programming*, 3-chi nashr. Bu kitobda iteratorlar, generatorlar va funksional yondashuvlar haqida bob bor. Packt
3. Luciano Ramalho. *Fluent Python*, 2-chi nashr. Bu kitobda "Iterables, Iterators, and Generators" bobida chuqur tushuntirishlar keltirilgan. O'Reilly Media
4. Julien Danjou va boshqa mualliflar. *Powerful Python* (O'Reilly) — "Scaling with Generators" bobiga ega. O'Reilly Media
5. Heather CS, Fast Lane to Python. Generatorlar va iteratorlar bo'yicha amaliy misollar va tushuntirishlar bor. heather.cs.ucdavis.edu
6. GeeksforGeeks maqolalari: "Using Generators for substantial memory savings in Python" — generatorlarning xotira tejamkorligi haqida. geeksforgeeks.org
7. MoldStud Research Team: "Understanding Python Generators - Your Ultimate Guide to Efficient Iteration". moldstud.com