

СУЧАСНІ ТЕХНОЛОГІЇ ПРОГРАМУВАННЯ

**122 «Комп'ютерні науки»
КН-19**

2010 / 2021 навчальний рік

PYTHON # 10

1. Перевантаження операторів
2. Об'єкт, що ітерується
3. Ітератор
4. Генератор

ПЕРЕВАНТАЖЕННЯ

Перевантаження операторів в Python:

- ❑ дозволяє класам брати участь в звичайних операціях,
- ❑ класи можуть перевантажувати всі оператори виразів,
- ❑ класи можуть також перевантажувати такі операції, як вивід, виклик функцій, звернення до атрибутів і ...
- ❑ перевантаження полягає в реалізації в класах методів із спеціальними іменами.

!!! якщо в класі визначено метод з спеціальним ім'ям, інтерпретатор буде автоматично викликати його при виконання відповідного методу екземпляру класу .

ПЕРЕВАНТАЖЕННЯ

Імена методів, що починаються і закінчуються двома символами підкреслення (__X__), мають спеціальне призначення.

Мова Python визначає фіксовані і незмінні імена методів для **кожної** з операцій. Перевантаження операторів реалізовано за рахунок створення методів зі цими спеціальними іменами для перехоплювання операцій.

Такі методи викликаються автоматично, коли екземпляр бере участь у вбудованих операціях.

МЕТОДИ ПЕРВАНТАЖЕННЯ

Метод	Перевантажує	Викликається
<code>__init__</code>	Конструктор	При створенні об'єкту
<code>__del__</code>	Деструктор	При знищенні об'єкту
<code>__repr__</code> <code>__str__</code>	Вивід, перетворення (строкове представлення)	<code>print()</code> , <code>repr()</code> , <code>str()</code> , <code>format ()</code>
<code>__len__</code>	Довжина	<code>len(x)</code>
<code>__bool__</code>	Перевірка логічного значення	
<code>__lt__</code> , <code>__gt__</code> , <code>__le__</code> , <code>__ge__</code> , <code>__eq__</code> , <code>__ne__</code> ,	Порівняння	$X < Y$, $X \leq Y$ $X > Y$, $X \geq Y$ $X == Y$, $X != Y$

ВБУДОВАНІ АРГУМЕНТИ (__init__)

Метод `__init__` є конструктором, який використовується для ініціалізації стану об'єкту.

Метод `__init__` завжди викликається при створенні нового екземпляру класу. Якщо метод `__init__` відсутнє в класі, інтерпретатор виконує пошук в суперкласі.

Важливо:

В методі `__init__` класу можливо викликати `__init__` суперкласу, за допомогою кваліфікованого ім'я.

ВБУДОВАНІ АРГУМЕНТИ (`__str__` , `__repr__`)

Строкове представлення об'єктів за замовчуванням має не містить корисної інформації і нелегко сприймається. Методи `__repr__` , `__str__` дозволяють визначити більш легкий для читання формат виведення екземплярів об'єктів.

Методи повертають строкове представлення екземпляру, завжди використовується для перетворення об'єкта *self.data* в рядок

Метод `__repr__` використовується всюди, за винятком функцій `print()` і `str()`, якщо визначено метод `__str__`. Якщо метод `__str__` відсутен, операції виведення використовуватимуть метод `__repr__`, але не навпаки.

АРИФМЕТИКО-ЛОГІЧНІ

Метод	Перевантажує	Викликається
<code>__add__</code> , <code>__radd__</code> , <code>__iadd__</code>	Оператор +	Додавання
<code>__sub__</code>	Оператор -	Віднімання
<code>__mul__</code>	Оператор *	Множення
<code>__truediv__</code>	Оператор /	Ділення
<code>__floordiv__</code>	Оператор //	Цілочислове ділення
<code>__mod__</code>	Оператор %	Залишок ділення
<code>__divmod__</code>	Оператор <code>divmod(X,Y)</code>	Частка ділення
<code>__pow__</code>	Оператор **	ступінь
<code>__and__</code>	Оператор &	«ТА»
<code>__or__</code>	Оператор ^	«АБО»
<code>__xor__</code>	Оператор	«Виняткове АБО»

ІТЕРАТИВНІСТЬ. ВИЗНАЧЕННЯ

Ітеруємий об'єкт (об'єкт, що ітерується, iterable, iterable object) - об'єкт, який має метод `__iter__()`, який повертає відповідний об'єкт - ітератор.

Ітератор (iterator) - об'єкт, який повертається методом `__iter__()` і має метод `__next__()`, що витягує (видає) наступний елемент контейнеру. При цьому цей елемент уже не міститься в ітераторі - ітератор в кінцевому підсумку спустошується і вертає помилку **StopIteration**.

ІТЕРАТИВНІСТЬ. ВИЗНАЧЕННЯ

Протокол ітерацій - об'єкт реалізує метод `_next_`, який вертає наступне значення послідовності і генерує виключну ситуацію *StopIteration*



ІТЕРАТИВНІСТЬ

Перевірка: є об'єкт ітеруємим ?

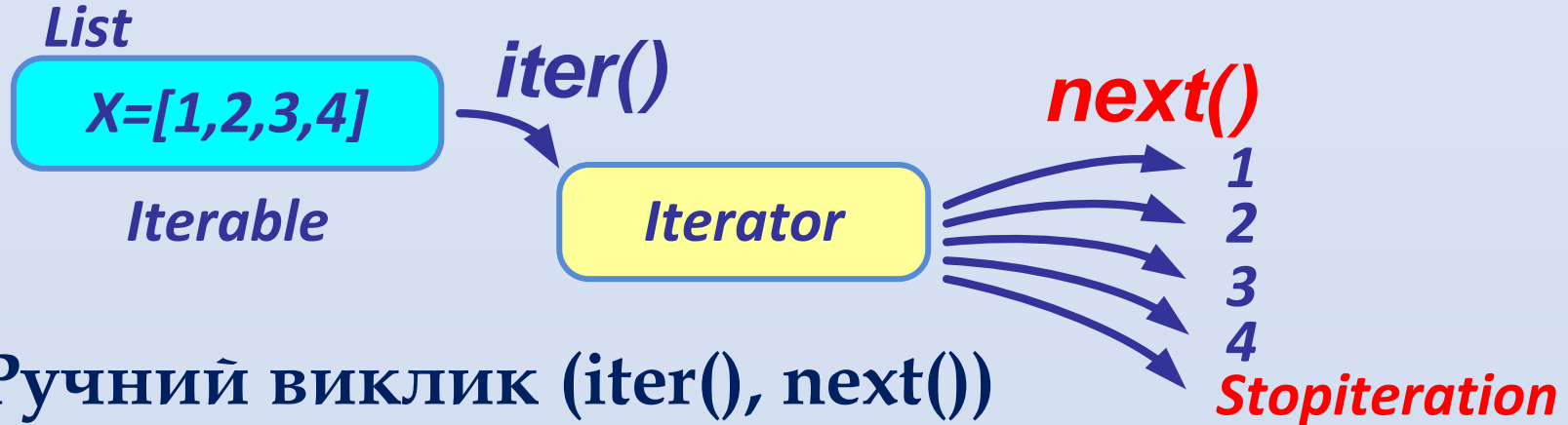
`hasattr(object , '__iter__')`

Приклад :

`hasattr(str , '__iter__')` → *True*

`hasattr(bool , '__iter__')` → *False*

ІТЕРАТОР



- Ручний виклик (`iter()`, `next()`)

- Ручний виклик WHILE

- Автоматичний виклик FOR
`for item in X : print (item)`

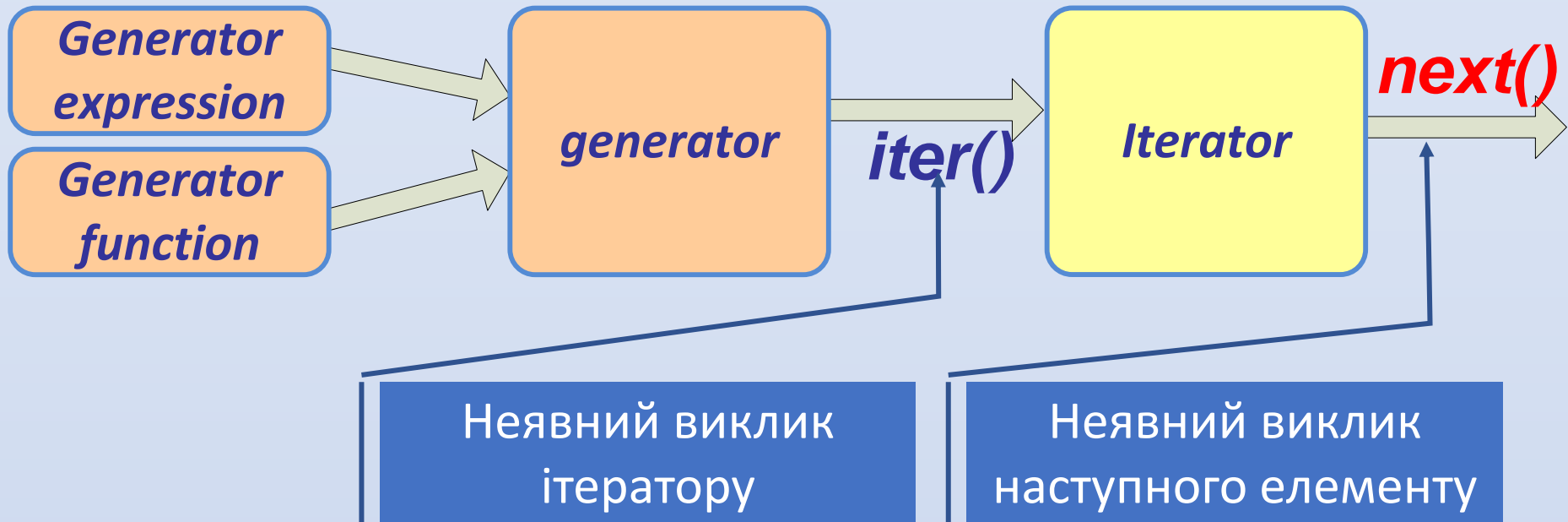
→1

→2

→3

→4

ГЕНЕРАТОР



Генератор створює методи **`__iter__`**, **`__next__`** автоматично!

Два типи генераторів:

1. **generator expression** - генератор - вираз
2. **generator function** - генератор - функція

ІТЕРАТИВНІСТЬ. ВИЗНАЧЕННЯ

Генератор (generator, generator expression) - спеціальний клас функцій, який дозволяє легко створювати свої ітератори.

На відміну від звичайних функцій, генератор не просто повертає значення і завершує роботу, а повертає **ітератор**, який віддає елементи по одному.

Генератор витягує (видає) значення. При цьому значення повертаються за запитом, і після повернення одного значення виконання функції - генератора **припиняється** до запиту наступного значення.

Між запитами генератор зберігає свій стан.

ГЕНЕРАТОР ФУНКЦІЯ

Головною особливістю генератора є повернення елементів на вимогу.

Генератор – функція забезпечує зручний спосіб реалізації протоколу ітерацій. **Генератор** - це ітерабельний об'єкт, створений за допомогою функції із **yield** інструкцією виходу.

Відмінність:

Звичайна функція припиняє своє виконання, коли вона виконує інструкцію виходу **return**.

Функція з інструкцією виходу **yield** зберігає свій стан, який може бути «підхоплений» наступного разу, коли викликається.

ГЕНЕРАТОР ВИРАЗ

Генератор-вираз – спрощений засіб створення генератору (порівняно з генератором – функцією).

Генератор – вираз дозволяє створювати генератор «в польоті» без використання інструкції виходу **yield**.

Генератор – вираз може бути записано з використанням синтаксису, схожого на синтаксис компоновки словника, але не в квадратних, а в круглих дужках.

ІНСТРУКЦІЯ FOR ... IN ...

```
for target_list in expression_list : suite  
    [else : suite]
```

Suite – група (блок) інструкцій

expression_list - об'єкт, що ітерується. Ітератор створюється після обчислення *expression_list*.

Suite виконується послідовно для кожного елемента що вертає ітератор. Коли елементи послідовності вичерпано (ітератор вертає *StopIteration*, виконується *suite* в виразі [*else* : *suite*] (якщо присутнє) і виконання циклу завершується.

Рекомендована ЛІТЕРАТУРА

- **Програмування числових методів мовою Python:** підруч. / А. В. Анісімов, А. Ю. Дорошенко, С. Д. Погорілий, Я. Ю. Дорогий ; за ред. А. В. Анісімова. – К. : Видавничо-поліграфічний центр "Київський університет", 2014. – 640 с.
- **Програмування числових методів мовою Python:** навч. посіб. / А. Ю. Дорошенко, С. Д. Погорілий, Я. Ю. Дорогий, Є. В. Глушко ; за ред. А. В. Анісімова. – К. : Видавничо-поліграфічний центр "Київський університет", 2013. – 463 с.
- **Основи програмування Python:** Підручник для студ. спеціальності 122 «Компютерні науки» / А.В.Яковенко; КПІ.- Київ: КПІ, 2018 . – 195 с.
- **Лутц М.** Изучаем Python, 4-е издание. - СПб.: Символ-Плюс. 2011.- 1280 с.: ил.

Контрольні запитання

- Наведіть переваги використання перевантаження операторів.
- Поясніть призначення методів `__str__`, `__repr__` та надайте приклади їх застосування.
- Поясніть призначення методу `__add__` та надайте приклади його перевантаження.
- Надайте визначення об'єкта, що ітерується, ітератора, генератора.
- Поясніть протокол ітерації в мові Python. Поясніть призначення методів `__iter__`, `__next__`.
- Наведіть приклад створення класу, що ітерується.
- Надайте визначення генератора – функції та генератора – виразу, поясніть їх відмінності. Наведіть приклади.

The END
Mod 1. Lec 09.