Елементи функціонального програмування

№ уроку:	1	**Курс:**	Python Advanced

Засоби навчання:	Python; інтегроване середовище розробки (PyCharm або Microsoft Visual Studio + Python Tools for Visual Studio)

Огляд, мета та призначення уроку

Після завершення уроку учні матимуть уявлення про основи парадигми функціонального програмування, навчаться використовувати деякі її принципи в програмах на Python (наприклад, створювати свої декоратори, які, по суті, є функціями вищого порядку), навчаться використовувати лямбда-вирази, стандартні функції та модулі, які пов'язані з функціональним програмуванням.

Вивчивши матеріал цього заняття, учень зможе:

- Мати уявлення про основи парадигми функціонального програмування.
- Використовувати її принципи в програмах Python.
- Використовувати функції як об'єкти першого класу.
- Використовувати лямбда-вирази.
- Створювати функції вищого порядку та власні декоратори.
- Використовувати функції filter, map і reduce.
- Використовувати модулі functools, operator та itertools.

Зміст уроку

1. Функції як об'єкти першого класу (first-class citizens).
2. Лямбда-вирази.
3. Замикання.
4. Функції вищого порядку, каррування функцій.
5. Декоратори.
6. Функції filter, map і reduce.
7. Модулі functools, operator та itertools.

Резюме

У курсі Python Essential описано об'єктно орієнтовану парадигму програмування та згадується, що є безліч менш популярних парадигм, які мають своє застосування. Одна з таких – **функціональна парадигма**. Зауважимо, що функціональна парадигма досить популярна та є конкурентом ООП.

Функціональне програмування (ФП) – парадигма програмування, у якій обчислення описуються як рішення функцій у математичному розумінні.

ФП передбачає використання обчислення результатів функцій від вихідних даних. Стан програми явно не зберігається. Відповідно стан не змінюється, на відміну від імперативного, де однією з базових концепцій є змінна, що зберігає своє значення й дає змогу змінювати його відповідно до виконання алгоритму.

На практиці відмінність математичної функції від поняття «функції» в імперативному програмуванні в тому, що **імперативні функції** можуть спиратися не тільки на аргументи, а й на стан зовнішніх щодо функції змінних, а також мати побічні ефекти та змінювати стан зовнішніх змінних.

В **імперативному програмуванні** під час виклику однієї й тієї ж функції з однаковими параметрами, але на різних етапах виконання алгоритму **можна отримати різні дані** на виході через вплив стану змінних на функцію.

CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

У функціональній мові під час виклику функції з тими самими аргументами ми **завжди матимемо однаковий результат**. Вихідні дані залежать тільки від вхідних. Це дає змогу середовищам виконання програм функціональними мовами кешувати результати функцій і викликати їх у порядку, що не визначається алгоритмом і розпаралелювати їх без будь-яких додаткових дій із боку програміста.

Python частково підтримує парадигму функціонального програмування та дає змогу писати код у функціональному стилі. Окрім цього, у ньому є певні можливості, які характерні для функціональних мов (спискові включення, лямбда-функції, функції вищого порядку тощо).

Об'єктами першого класу *(англ. first-class object, first-class entity, first-class citizen)* у контексті конкретної мови програмування називаються сутності, які можуть бути передані як параметр, повернуті з функції та присвоєні змінній.

Об'єкт називають *«об'єктом першого класу»,* якщо він:

- може бути збережений у змінній чи структурах даних;
- може бути переданий у функцію як аргумент;
- може бути повернений із функції як результат;
- може бути створений під час виконання програми;
- незалежний від назви.

Термін «об'єкт» використовується тут у загальному розумінні та не обмежується об'єктами мови програмування.

У Python, як і функціональних мовах, функції **є об'єктами першого класу**.

Звичайне оголошення функції Python є оператором def. Однак під час написання коду у функціональному стилі часто буває зручною можливість оголосити анонімну функцію усередині виразу. У Python є така можливість: вона реалізується за допомогою лямбда-виразів.

В анонімних функціях Python є обмеження: вони можуть складатися лише з одного виразу. У деяких мовах такого обмеження немає.

Замикання (*англ. closure*) у програмуванні – функція, у тілі якої є посилання на змінні, що оголошені поза тілом цієї функції в навколишньому коді та не є її параметрами.

Іншими словами, замикання – можливість функції, яка посилається на змінну зі свого контексту.

Замикання, як і екземпляр об'єкта, є способом представлення функціональності та даних, що пов'язані й упаковані разом. **Замикання** – це особливий різновид функції. Воно визначене в тілі іншої функції та створюється щоразу під час її виконання.

Синтаксично це виглядає як функція, що повністю розташована в тілі іншої функції. Водночас вкладена внутрішня функція містить посилання на локальні змінні зовнішньої функції. Щоразу під час виконання зовнішньої функції відбувається створення нового екземпляра внутрішньої функції з новими посиланнями на змінні зовнішньої функції.

У разі замикання посилання на змінні зовнішньої функції дійсні всередині вкладеної функції доти, доки працює вкладена функція, навіть якщо зовнішня функція закінчила роботу, і змінні вийшли з області видимості.

У цьому прикладі ми виконали функцію make_closure. Здавалося б, вона виконалася, і все, що було створено в ній, перестало існувати. Але лексичне оточення батьківської функції доступне дочірньою функцією і, відповідно, може отримати доступ до батьківських змінних.

CyberBionic systematics

CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

Замикання пов'язує код функції з її лексичним оточенням (місцем, де вона визначена в коді). Лексичні змінні замикання відрізняються від глобальних змінних тим, що вони займають глобальний простір імен. Від змінних в об'єктах вони відрізняються тим, що прив'язані до функцій, а не об'єктів.

У Python будь-які функції (зокрема й лямбда-вирази), які оголошені всередині інших функцій, є повноцінними замиканнями.

Функція вищого порядку – функція, яка приймає як аргументи інші функції або повертає іншу функцію як результат. Основна ідея полягає в тому, що функції мають той самий статус, що й інші об’єкти даних.

Каррування, або Каррінг (*англ. currying*) в інформатиці – перетворення функції від багатьох аргументів на функцію, яка бере свої аргументи по одному. Це перетворення було запроваджено М. Шейнфінкелем і Г. Фреге й отримало свою назву на честь Г. Каррі.

Є такий патерн проєктування, як-от **декоратор**. Це структурний шаблон проєктування, який призначений для динамічного під'єднання додаткової поведінки до об'єкта. Шаблон декоратор пропонує гнучку альтернативу практиці створення підкласів для розширення функціональності.

Декоратор Python – функція, яка приймає іншу функцію (або клас) і повертає нову функцію (або клас). Цей механізм дає змогу обернути іншу функцію для розширення її функціональності без безпосередньої зміни її коду.

Часто потрібно, щоби декоратор приймав ще якісь параметри, окрім об'єкта, що модифікується. У такому випадку створюється функція, яка створює та повертає декоратор, а під час застосування декоратора замість вказівки імені функції-декоратора ця функція викликається.

Трьома класичними функціями вищого порядку, що з'явилися ще в мові програмування Lisp, які приймають функцію та послідовність, є **map, filter та reduce**.

Функція **map** застосовує функцію до кожного елемента послідовності. У Python 2 повертає список, а в Python 3 – об'єкт-ітератор.

Функція **filter** залишає лише ті елементи послідовності, для яких задана функція істинна. У Python 2 повертає список, а у Python 3 – об'єкт-ітератор.

Функція **reduce** (у Python 2 вбудована, а в Python 3 розташована в модулі functools) приймає функцію від двох аргументів, послідовність і опціональне початкове значення та обчислює згортку (fold) послідовності як результат послідовного застосування цієї функції до поточного значення (так званого акумулятора) та наступного елемента послідовності.

Модуль **functools** містить велику кількість стандартних функцій вищого порядку. Серед них особливо корисні:

- reduce – розглянута вище;
- lru_cache – декоратор, що кешує значення функцій, які не змінюють свій результат за незмінних аргументів; корисний для кешування даних, мемоізації (збереження результатів для повернення без обчислення функції) значень рекурсивних функцій (наприклад, такого типу, як-от функція обчислення n-го числа Фібоначчі) тощо;
- partial – часткове застосування функції (виклик функції з меншою кількістю аргументів, ніж вона очікує, та отримання функції, яка приймає параметри, що залишилися).

Модуль **itertools** містить функції для роботи з ітераторами та створення ітераторів. Деякі з його функцій:

- product() – декартів добуток ітераторів (для уникнення вкладених циклів for);
- permutations() – генерація перестановок;
- combinations() – генерація сполучень;
- combinations_with_replacement() – генерація розміщень;
- chain() – з'єднання кількох ітераторів в один;

CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

- takewhile() – отримання значень послідовності, поки значення функції-предикату для її елементів істинне;
- dropwhile() – отримання значень послідовності починаючи з елемента, для якого значення функції-предиката перестане бути істинним.

Модуль operator містить функції, які відповідають стандартним операторам. У такий спосіб замість lambda x, y: x + y можна використовувати вже готову функцію operator.add тощо.

Закріплення матеріалу

- Що таке функціональне програмування?
- Що таке об'єкт першого класу?
- Що таке лямбда-вираз?
- Що таке замикання?
- Що таке функція найвищого порядку?
- Що таке каррування?
- Що таке декоратор?
- Що робить функція map?
- Що робить функція filter?
- Що робить функція reduce?
- Що таке часткове застосування функції?

Додаткове завдання

Створіть звичайну функцію множення двох чисел. Частково застосуйте її до одного аргументу. Створіть каррувану функцію множення двох чисел. Частково застосуйте її до одного аргументу.

Самостійна діяльність учня

Завдання 1
Ще раз розберіть усі приклади до уроку, повторіть теорію та ознайомтеся з документацією щодо розглянутих модулів.

Завдання 2
Створіть список цілих чисел. Отримайте список квадратів непарних чисел із цього списку.

Завдання 3
Створіть функцію-генератор чисел Фібоначчі. Застосуйте до неї декоратор, який залишатиме в послідовності лише парні числа.

Рекомендовані ресурси

Документація Python
https://docs.python.org/3/library/functions.html#filter
https://docs.python.org/3/library/functions.html#map
https://docs.python.org/2/library/functions.html#reduce
https://docs.python.org/3/library/functools.html
https://docs.python.org/3/library/itertools.html
https://docs.python.org/3/library/operator.html
https://www.python.org/dev/peps/pep-0318/

Статті у Вікіпедії про ключові поняття, які розглянуті на цьому уроці
https://uk.wikipedia.org/wiki/Функційне_програмування
https://uk.wikipedia.org/wiki/Об'єкт_першого_класу
https://uk.wikipedia.org/wiki/ Замикання_(програмування)
https://uk.wikipedia.org/wiki/Функція_вищого_порядку

CyberBionic systematics
CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine
Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

https://uk.wikipedia.org/wiki/Каррінг_(інформатика)
https://uk.wikipedia.org/wiki/Декоратор_(шаблон_проєктування)
https://uk.wikipedia.org/wiki/Функційне_програмування

CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com