Інтроспекція та рефлексія

№ уроку:	5	**Курс:**	Python Essential

Засоби навчання:	Python 3; 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 основні принципи програмування: інтроспекції та рефлексії, а також застосовувати їх на практиці.

Вивчивши матеріал цього заняття, учень зможе:

- Мати уявлення про основні принципи програмування: інтроспекцію та рефлексію.
- Мати уявлення про основні атрибути у мові Python, які допоможуть досліджувати стан об’єктів та класів.
- Використовувати інтроспекцію та рефлексію практично під час виконання програм.

Зміст уроку

- Що таке інтроспекція?
- Застосування інтроспекції на практиці
- Що таке рефлексія?
- Застосування інтроспекції на практиці

Резюме

- **Інтроспекція** (англ. type introspection) — можливість отримати тип та структуру об'єкта під час виконання програми. Особливе значення має у мові Objective C, проте є майже у всіх мовах, які дають змогу маніпулювати типами об'єктів як об'єктами першого класу; серед мов, що підтримують інтроспекцію: C++ (з RTTI), Go, Java, Kotlin, JavaScript, Perl, Ruby, Smalltalk; у PHP і Python інтроспекція інтегрована в саму мову.
- Інтроспекція може бути використана для реалізації ad-hoc-поліморфізму.
- У Python інтроспекція може бути функціонально реалізована за допомогою вбудованих методів type() і dir() або вбудованого модуля inspect, або йти безпосередньо від імені об'єкта за допомогою вбудованих атрибутів __class__ та __dict__. Користуватися інтроспекцією в Python особливо зручно завдяки парадигмі «все є об'єктом». Будь-яка сутність, бувши об'єктом, має метадані (дані про об'єкт), які називаються атрибутами, і пов'язані з цією сутністю функціональності, які називаються методами. У Python новий клас за замовчуванням є сам собою об'єктом метакласу type.
- Python підтримує повну інтроспекцію (віддзеркалення) часу виконання, у тому числі інтроспекцію типу (type introspection (англ.)). Це означає, що для будь-якого об'єкта можна отримати всю інформацію про його внутрішню структуру та середовище виконання. Можливості інтроспекції можна умовно розділити на дві групи: стандартні (описані в документації з мови (англ.)) та нестандартні (характерні для конкретної реалізації мови, наприклад, CPython).
- Необхідні для інтроспекції дані зберігаються у спеціальних атрибутах. Так, наприклад, отримати всі атрибути більшості об'єктів можна зі спеціального атрибута - словника (або об'єкта, що надає dict інтерфейс) __dict__.
- Інтроспекція дозволяє вивчати атрибути об'єкта під час виконання програми, а рефлексія — маніпулювати ними.
- Найбільш важливі атрибути та функції мови для роботи з інтроспекцією наступні:
  __class__
  __base__
  __bases__

CyberBionic systematics
CyberBionic Systematics ® 2015
19 Mariny Raskovoy Str., 5 floor
Kyiv, Ukraine
t. +380 (44) 361-8473
E-mail: edu@cbsystematics.com
Site:www.edu.cbsystematics.com

__dict__
dir()
type()
type(», (), {})
isinstance()
issubclass()

- **Рефлексія** (англ. reflection) – це здатність комп'ютерної програми вивчати і модифікувати свою структуру та поведінку (значення, мета-дані, властивості та функції) під час виконання. Вона дозволяє викликати методи об'єктів, створювати нові об'єкти, модифікувати їх, навіть не знаючи імен інтерфейсів, полів, методів під час компіляції. Через таку природу рефлексії її важче реалізувати в статично типізованих мовах, оскільки помилки типізації виникають під час компіляції, а не виконання програми (докладніше про це тут). Тим не менш, вона можлива, адже такі мови, як Java, C# та інші допускають використання як інтроспекції, так і рефлексії (але не C++ він дозволяє використовувати лише інтроспекцію). З тієї ж причини рефлексію простіше реалізувати в мовах, що інтерпретуються, оскільки коли функції, об'єкти та інші структури даних створюються і викликаються під час роботи програми, використовується якась система розподілу пам'яті. Інтерпретовані мови зазвичай надають таку систему за замовчуванням, а компілюваних знадобиться додатковий компілятор і інтерпретатор, який стежить за коректністю рефлексії.
- Рефлексивно-орієнтоване програмування або рефлексивне програмування — функціональне розширення парадигми об'єктно-орієнтованого програмування.
- Рефлексивно-орієнтоване програмування включає самоперевірку, самодифікацію та самоклонування. Тим не менш, головна перевага рефлексивно-орієнтованої парадигми полягає в динамічній модифікації програми, яка може бути визначена та виконана під час роботи програми. Деякі імперативні підходи, наприклад, процедурна та об'єктно-орієнтована парадигми програмування, вказують, що існує чітка зумовлена послідовність операцій обробки даних.
- Парадигма рефлексивно-орієнтованого програмування додає можливість динамічної модифікації програмних інструкцій під час роботи та їх виклику в модифікованому вигляді. Тобто програмна архітектура сама визначає, що саме можна робити під час роботи, виходячи з даних, сервісів та специфічних операцій.
- Рефлексія може використовуватись для спостереження та зміни програми під час виконання. Рефлексивний компонент програми може спостерігати за виконанням певної ділянки коду та змінювати себе для досягнення бажаної мети. Модифікація виконується під час виконання програми шляхом динамічної зміни коду.
- Рефлексію можна застосовувати і для динамічної адаптації програми до різних ситуацій. Наприклад, розглянемо програму, що використовує два різних класи X та Y для виконання аналогічних операцій. Без рефлексії у коді програми методи класів X та Y будуть викликатися явно. Якщо програма спроектована із застосуванням рефлексивно-орієнтованої парадигми програмування, деяка ділянка коду не міститиме явних викликів методів класів X та Y; програма виконає цю ділянку двічі: спочатку класу X, потім класу Y.
- Прикладом, що прояснює переваги рефлексії, може бути серіалізація об'єкта в JSON. Без рефлексії необхідно було б явно вказувати всі імена полів класу і посилатися на їх значення для серіалізації. Але рефлексія дозволяє програмі самій визначити всі наявні поля та отримати їх текстові імена. Таким чином, серіалізація стає доступною для будь-якого об'єкта без написання зайвого коду.
- Програми, написані мовами програмування, що підтримують рефлексію, мають додаткові можливості, реалізація яких мовами низького рівня є скрутною. Перерахуємо деякі з них:
  • пошук та модифікація конструкцій вихідного коду (блоків, класів, методів, інтерфейсів (протоколів) тощо) як об'єктів першого класу під час виконання;
  • зміна імен класів та функцій під час виконання;
  • аналіз та виконання рядків коду, що надходять ззовні;
  • створення інтерпретатору байт-коду нової мови.
- Найбільш важливі атрибути та функції мови для роботи з рефлексією наступні:
  hasattr()

CyberBionic systematics
CyberBionic Systematics ® 2015
19 Mariny Raskovoy Str., 5 floor
Kyiv, Ukraine
t. +380 (44) 361-8473
E-mail: edu@cbsystematics.com
Site:www.edu.cbsystematics.com

delattr()
setattr()
getattr()

- Для зручності отримання інтроспективної інформації в Python є модуль inspect. Модуль **inspect** надає кілька корисних функцій, що допомагають отримати інформацію про живі об'єкти, такі як модулі, класи, методи, функції, трасування, кадри та об'єкти коду. Наприклад, він може допомогти вам вивчити зміст класу, отримати вихідний код методу, витягти та відформатувати список аргументів для функції або отримати всю інформацію, необхідну для відображення детального трасування.
- Цей модуль надає чотири основні види послуг: перевірка типів, отримання вихідного коду, перевірка класів та функцій та перевірка стека інтерпретатору.
- Функція **getmembers()** витягує члени об'єкта, наприклад клас або модуль. Функції, імена яких починаються з «is», переважно надаються як зручний вибір другого аргументу **getmembers()**. Вони також допоможуть вам визначити, коли ви можете очікувати, щоб знайти такі особливі атрибути:

Тип	Атрибут	Опис
module	__doc__	рядок документації
	__file__	ім'я файлу (відсутнє для вбудованих модулів)
class	__doc__	рядок документації
	__name__	ім'я, з яким було визначено цей клас
	__qualname__	складове ім'я
	__module__	ім'я модуля, в якому було визначено цей клас
method	__doc__	рядок документації
	__name__	ім'я, за допомогою якого було визначено цей метод
	__qualname__	складове ім'я
	__func__	об'єкт функції, що містить реалізацію методу
	__self__	сутність, з якою пов'язаний цей метод або None
	__module__	ім'я модуля, в якому було визначено цей метод
function	__doc__	рядок документації
	__name__	ім'я, з яким було визначено цю функцію
	__qualname__	складове ім'я
	__code__	кодовий об'єкт, що містить скомпільовану функцію байт-коду
	__defaults__	кортеж будь-якого значення за промовчанням для позиційних чи ключових параметрів
	__kwdefaults__	відображення будь-якого значення за замовчуванням лише для ключових параметрів
	__globals__	глобальний простір імен, у якому було визначено цю функцію
	__annotations__	відображення імен параметрів в інструкції; "return" ключ зарезервований для анотації, що повертається.
	__module__	ім'я модуля, в якому було визначено цю функцію
traceback	tb_frame	об'єкт кадру на цьому рівні
	tb_lasti	індекс останньої спроби виконання інструкції у байт-коді
	tb_lineno	поточний номер рядка у вихідному Python коді
	tb_next	наступний внутрішній об'єкт трейсбека (викликається цим рівнем)

CyberBionic systematics
CyberBionic Systematics ® 2015
19 Mariny Raskovoy Str., 5 floor
Kyiv, Ukraine
t. +380 (44) 361-8473
E-mail: edu@cbsystematics.com
Site:www.edu.cbsystematics.com

Тип	Атрибут	Опис
frame	f_back	наступний об'єкт зовнішнього кадру (об'єкт цього кадру, що викликає)
	f_builtins	вбудований простір імен, видимий цим кадром
	f_code	об'єкт коду виконуваний у цьому кадрі
	f_globals	глобальний простір імен, видимий цим кадром
	f_lasti	індекс останньої спроби виконання команди у байт-коді
	f_lineno	поточний номер рядка у вихідному Python коді
	f_locals	локальний простір імен, видимий цим кадром
	f_trace	відстеження функції для цього кадру або None
code	co_argcount	кількість аргументів (не включаючи лише ключові аргументи, * або ** args)
	co_code	рядок вихідного скомпілюваного байт-коду
	co_cellvars	кортеж імен змінних осередку (на який посилається області видимості)
	co_consts	кортеж констант використовується в байт-коді
	co_filename	ім'я файлу, в якому було створено цей об'єкт коду
	co_firstlineno	номер першого рядка у вихідному Python коді
	co_flags	бітова карта прапорів CO_*
	co_lnotab	кодований відображення номерів рядків на індекси байт-коду
	co_freevars	кортеж імен вільних змінних (на які посилається замикання функції)
	co_posonlyargcount	кількість лише позиційних аргументів
	co_kwonlyargcount	число лише ключових аргументів (не включаючи ** arg)
	co_name	ім'я, з яким було визначено цей об'єкт коду
	co_names	кортеж імен локальних змінних
	co_nlocals	число локальних змінних
	co_stacksize	необхідний простір стека віртуальної машини
	co_varnames	кортеж імен аргументів та локальних змінних
generator	__name__	ім'я
	__qualname__	складове ім'я
	gi_frame	фрейм
	gi_running	чи працює генератор?
	gi_code	код
	gi_yieldfrom	об'єкт, що ітерується yield from або None
coroutine	__name__	ім'я
	__qualname__	складове ім'я
	cr_await	очікуваний об'єкт або None
	cr_frame	фрейм
	cr_running	чи працює корутина?
	cr_code	код
	cr_origin	де було створено корутину чи None. sys.set_coroutine_origin_tracking_depth()

CyberBionic systematics

CyberBionic Systematics ® 2015
19 Mariny Raskovoy Str., 5 floor
Kyiv, Ukraine

t. +380 (44) 361-8473
E-mail: edu@cbsystematics.com
Site:www.edu.cbsystematics.com

Тип	Атрибут	Опис
builtin	__doc__	рядок документації
	__name__	вихідне ім'я цієї функції чи методу
	__qualname__	складове ім'я
	__self__	сутність, з яким пов'язаний метод або None

Закріплення матеріалу

- Що таке інтроспекція?
- Що таке рефлексія?
- Які основні атрибути дозволяють використовувати у мові Python інтроспекцію та рефлексію?
- Який модуль допоможе дізнатися стан об’єкта?
- Які основні інструменти є в цьому модулі?

Додаткові завдання

Завдання 1
Використовуючи код завдання 2 надрукуйте у терміналі інформацію, яка міститься у класах Contact та UpdateContact та їх екземплярах. Видаліть атрибут job, і знову надрукуйте стан класів та їх екземплярів. Порівняйте їх. Зробіть відповідні висновки.

Завдання 2
Використовуючи код завдання 2 надрукуйте у терміналі всі методи, які містяться у класі Contact та UpdateContact.

Самостійна діяльність учня

Завдання 1
Вивчити роботу інструментів, які розглядалися на занятті.

Завдання 2
Створити клас Contact з полями surname, name, age, mob_phone, email. Додати методи get_contact, sent_message. Створити клас-нащадок UpdateContact з полями surname, name, age, mob_phone, email, job. Додати методи get_message. Створити екземпляри класів та дослідити стан об'єктів за допомогою атрибутів: __dict__, __base__, __bases__. Роздрукувати інформацію на екрані.

Завдання 3
Використовуючи код з завдання 2, використати функції hasattr(), getattr(), setattr(), delattr(). Застосувати ці функції до кожного з атрибутів класів, подивитися до чого це призводить.

Завдання 4
Використовуючи код з завдання 2, створити 2 екземпляри обох класів. Використати функції isinstance() – для перевірки екземплярів класу (за яким класом створені) та issubclass() – для перевірки і визначення класу-нащадка.

Рекомендовані ресурси

Документація Python 3
Інформація про механізм огляду об'єктів
https://docs.python.org/3/library/inspect.html?highlight=introspection

CyberBionic Systematics ® 2015
19 Mariny Raskovoy Str., 5 floor
Kyiv, Ukraine
t. +380 (44) 361-8473
E-mail: edu@cbsystematics.com
Site:www.edu.cbsystematics.com