ООП – Успадкування та абстракція

№ уроку: 2 **Курс:** Python Essential

Засоби навчання: PyCharm

Огляд, мета та призначення уроку

Після завершення уроку учні розширять своє уявлення про парадигму об'єктно-орієнтованого програмування та його реалізацію у мові Python, зможуть розуміти та використовувати принципи успадкування.

Вивчивши матеріал цього заняття, учень зможе:

- Розуміти механізми успадкування та множинного успадкування.
- Створювати ієрархії класів у програмах на Python.
- Розуміти, що таке MRO.

Зміст уроку

1. Що таке успадкування?
2. Реалізація успадкування в Python.
3. Що таке множинне успадкування?
4. Реалізація множинного успадкування у Python.
5. Як працює успадкування у Python, що таке MRO?
6. Що таке super?
7. Декоратори @staticmethod і @classmethod
8. UML-діаграми.
9. Реалізація у Python.

Резюме

Успадкування – один з основних принципів ООП. Цей механізм дає вам змогу створити розширений клас інших класів.
Підклас успадковує атрибути та методи з батьківського класу. Він також може перевизначати (**override**) методи батьківського класу. Наприклад, якщо підклас не визначає свій ініціалізатор, він успадкує його від батьківського класу за замовчуванням.
Успадкування в об'єктно-орієнтованому програмуванні схоже на успадкування в реальному житті, коли дитина успадковує характеристики батьків і створює власні характеристики.
Основна ідея успадкування в об'єктно-орієнтованому програмуванні: **клас може успадковувати характеристики іншого класу**. Клас, який успадковує інший клас, називається **дочірнім** класом. Клас, який дає спадщину, називається **батьківським** або **основним**.
Наприклад, кошеня може успадковувати колір шкіри й очей свого батька. Але так само у кошеня може з'явитися власна характеристика: особлива цятка на шерсті. Отже, успадкування – це процес «запозичення» деяких характеристик батька з можливістю їхнього перевизначити чи додати нові.

Що таке множинне успадкування?
Хто такий пегас: кінь чи птах? І те, й інше: тіло коня, крила птаха. Тобто, якщо у нас є клас коня та клас птиці, нам потрібно поєднати два класи в одному? Це можливо за допомогою **множинного успадкування** – успадкування характеристик одночасно від двох класів.
Разом із цим механізмом постає багато запитань і проблем. Наприклад, якщо у двох батьківських класах є однакові методи, із якого класу метод буде в дочірньому класі? Щоб дати відповідь на це запитання, нам треба заглибитися в механізм реалізації успадкування в Python.

Реалізація успадкування в Python
Розглянемо приклад успадкування в Python:

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

```
# example_1.py

class Cat:
 def __init__(self, name):
 self.name = name

 def say_meow(self):
 print(f"{self.name}: Meow!")

class Kitty(Cat):
 pass

my_cat = Cat("Black")
my_kitty = Kitty("Gray")

my_cat.say_meow()
my_kitty.say_meow()
```

Створено клас Cat, визначено ініціалізатор і метод say_meow. Створено клас Kitty та успадковано весь функціонал їхнього класу Cat. **Щоб успадкувати клас, потрібно вписати назву батьківського класу всередині дужок, які йдуть за назвою дочірнього класу**.

Далі створюється два об'єкти з класу Cat і Kitty. Kitty успадкував ініціалізатор із класу Cat. Тому ми можемо ініціалізувати Kitty так само як і Cat. Успадковані також методи: ми можемо викликати на об'єкті метод say_meow.
Додамо до класу Kitty власних методів:

```
# example_2.py

class Cat:
 def __init__(self, name):
 self.name = name

 def say_meow(self):
 print(f"{self.name}: Meow!")

class Kitty(Cat):
 def say_meow(self):
 print(f"{self.name}: Meow from kitty!")

 def bite_a_finger(self):
 print("Bite! :)")


my_cat = Cat("Black")
my_kitty = Kitty("Gray")

my_cat.say_meow()
my_kitty.say_meow()
my_kitty.bite_a_finger()
```

1. Перевизначено метод say_meow. Це означає, що навіть якщо ми успадковуємося від іншого класу з таким самим методом, більший пріоритет має наш метод.
2. Доданий метод bite_a_finger, якого немає у батьківському класі.
3. Ініціалізатор не перевизначений, тому клас Kitty, як і раніше, використовує ініціалізатор класу Cat.

Реалізація множинного успадкування в Python
Повернемося до прикладу з пегасом. Щоб попрактикуватися, рекомендую написати самостійно класи Bird і Horse. Клас Bird має мати метод fly, клас Horse – метод run. Додамо клас Pegas:

```
# example_3.py
```


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

```
class Bird:
 TYPE = "Bird"

 def fly(self):
 print(f"I am a {self.TYPE} and I can fly!")

class Horse:
 TYPE = "Horse"

 def run(self):
 print(f"I am a {self.TYPE} and I can run!")

class Pegas(Bird, Horse):
 pass

my_home_pegas = Pegas()
my_home_pegas.run()
my_home_pegas.fly()
```

Тепер наш пегас вміє і літати, і бігати! Нам просто потрібно вказати через кому два батьківські класи. Але якщо запустити код, ми побачимо таке:

```
# python example_3.py

I am a Bird and I can run!
I am a Bird and I can fly!
```

Ми жодного разу не вказали TYPE = "Pegas", тому логічно, що ми **не** бачимо **"I am a Pegas and I can run!"**. Чому саме Bird, а не Horse – ми поговоримо у темі MRO. Зараз виправимо цю неточність (спробуйте самостійно):

```
# example_4.py

class Bird:
 TYPE = "Bird"

 def fly(self):
 print(f"I am a {self.TYPE} and I can fly!")

class Horse:
 TYPE = "Horse"

 def run(self):
 print(f"I am a {self.TYPE} and I can run!")

class Pegas(Bird, Horse):
 TYPE = "Pegas"

my_home_pegas = Pegas()
my_home_pegas.run()
my_home_pegas.fly()
```

Ми додали до Pegas атрибут TYPE = "Pegas". Тепер все працює коректно.

Як працює успадкування у Python, що таке MRO?

MRO (Method Resolution Order) – це порядок, у якому методи шукаються в основному та батьківських класах. MRO використовує алгоритм C3 лінеаризації. За допомогою цього алгоритму Python вибудовує лінійний ланцюжок, як треба шукати метод у класі.

Насамперед Python шукає метод або атрибут у класі, до якого належить об'єкт. Після, за порядком MRO, Python шукає цей метод\атрибут у батьківських класах. Повернемося до першого прикладу та розглянемо його з боку MRO:

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

```
# example_1.py

class Cat:
 def __init__(self, name):
 self.name = name

 def say_meow(self):
 print(f"{self.name}: Meow!")

class Kitty(Cat):
 pass

my_cat = Cat("Black")
my_kitty = Kitty("Gray")

my_cat.say_meow()
my_kitty.say_meow()
```

my_kitty.say_meow() – ми викликаємо метод, якого немає у класі цього об'єкта. Але він є в класі батька, тому в Python ланцюжок пошуку методу виглядає так: (Kitty, Cat). Якщо словами: «Спочатку шукаємо метод у класі Kitty, якщо не знайдеш, шукай у Cat».

Розглянемо складніший приклад:

```
# example_4.py

class Bird:
 TYPE = "Bird"

 def fly(self):
 print(f"I am a {self.TYPE} and I can fly!")

class Horse:
 TYPE = "Horse"

 def run(self):
 print(f"I am a {self.TYPE} and I can run!")

class Pegas(Bird, Horse):
 TYPE = "Pegas"

my_home_pegas = Pegas()
my_home_pegas.run()
my_home_pegas.fly()
```

my_home_pegas.run() викликаємо метод, якого немає в пегасі. Як дізнатися, в якому класі Python шукатиме цей метод насамперед? Можна докладно вивчити алгоритм лінеаризації C3 або просто подивитися в атрибут класу __mro__:

```
print(Pegas.__mro__)# Result: (<class '__main__.Pegas'>, <class '__main__.Bird'>, <class
'__main__.Horse'>, <class 'object'>)
```

Це той самий ланцюжок, за яким Python шукає методи й атрибути. Спочатку Python шукає метод\атрибут у класі Bird, потім у Horse. Якщо класи поміняти місцями class Pegas(Bird, Horse):, ланцюжок зміниться: (<class '__main__.Pegas'>, <class '__main__.Horse'>, <class '__main__.Bird'>, <class 'object'>).

Що таке функція super()?

Іноді треба не перевизначити метод, а розширити його функціонал. Наприклад:

```
# example_5.py
```


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

```
class MathBase:
 def math_operation(self, number_1, number_2, operation):

 if operation == "-":
 return number_1 - number_2
 elif operation == "+":
 return number_1 + number_2

class MathUpgrated(MathBase):
 def math_operation(self, number_1, number_2, operation):
 if operation in ["+", "-"]:
 parent_class_object = super()
 result = parent_class_object.make_math_operation(number_1, number_2, operation)
 return result
 elif operation == "*":
 return number_1 * number_2
 elif operation == "/":
 return number_1 / number_2

my_math = MathUpgrated()

print(my_math.math_operation(4, 2, "+")) # ==> 6
print(my_math.math_operation(4, 2, "-")) # ==> 2
print(my_math.math_operation(4, 2, "/")) # ==> 2.0
print(my_math.math_operation(4, 2, "*")) # ==> 8
```

Є клас MathBase і метод math_operation, що вміє працювати з операціями + і -. Завдання класу MathUpgrated розширити функціонал методу math_operation операціями * та /.

У новому методі класу MathUpgrated можна продублювати код із базового класу, але копіювання коду в програмуванні — погана практика (методологія DRY). Це означає, що з нового методу потрібно зберегти можливість звернутися до батьківських методів. Для цього у Python є функція super(). Якщо її викликати всередині методу, ми отримаємо такий самий **об'єкт** як **self**, але з методами батьківського класу.

```
# Отримуємо self, але з методами батьківського класу - MathBase:
 parent_class_object = super()

 # Викликаємо метод "make_math_operation" на батьківському класі, передаючи всі необхідні
аргументи result = parent_class_object.make_math_operation(number_1, number_2, operation) #
Повертаємо результат return result# ...
```

Зверніть увагу: функція super() повертає об'єкт **наступного класу за MRO**.

Декоратор (англ. *Decorator*) — 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.

Декоратори – це прості функції, які приймають на вхід функцію. Найчастіше вони зображуються як «@my_decorator» над функцією, що декорується.

Статичний метод – це спосіб помістити функцію до класу, якщо вона логічно належить до цього класу. Статичний метод нічого не знає про клас, із якого його викликали.

Класовий метод навпаки знає, з якого класу його викликають. Він приймає неявний перший аргумент (зазвичай він має назву ***cls***), який містить клас, що викликають. Класові методи чудово підходять, коли потрібно врахувати ієрархію успадкування.

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

Абстракція в ООП — це використання лише тих характеристик об'єкта, які з достатньою точністю представляють його в цій системі. Основна ідея полягає в тому, щоб представити об'єкт мінімальним набором полів і методів та водночас із достатньою точністю для задачі, що розв'язується.

Це важливий інструмент ООП поряд із поліморфізмом, успадкуванням та інкапсуляцією.

Абстракція є основою об'єктно-орієнтованого програмування та дає змогу працювати з об'єктами, не вдаючись в особливості їхньої реалізації.

Абстракція даних — одне з найстаріших понять об'єктно-орієнтованого програмування, що виникло ще до його появи. Абстракція даних пов'язує тип даних, що лежить в основі, з набором операцій над ним (див. також абстрактний тип даних). Користувач типу даних немає прямого доступу до його реалізації, але може працювати з даними через наданий набір операцій. Перевага абстракції даних полягає у розподілі операцій над даними та внутрішньому поданні цих даних, що дає змогу змінювати реалізацію, не торкаючись користувачів типу даних.

Такий поділ може бути виражений через спеціальний «інтерфейс», який зосереджує опис усіх можливих застосувань програми.

Unified Modeling Language (UML) – уніфікована мова моделювання. Modeling — створення моделі, що описує об'єкт. Unified (універсальний, єдиний) підходить для широкого класу програмних систем, що проєктуються, різних областей застосунків, типів організацій, рівнів компетентності, розмірів проєктів.

UML описує об'єкт у єдиному заданому синтаксисі, тому де б ви не намалювали діаграму, її правила будуть зрозумілі всім, хто знайомий з цією графічною мовою — навіть в іншій країні.

Закріплення матеріалу

- Що таке успадкування?
- Що таке множинне успадкування?
- Як у Python вказати, що один клас успадковується від іншого класу? Від кількох інших класів?
- Що таке MRO?
- Як отримати доступ до методу базового класу, якщо він був перевизначений у цьому класі?
- Для чого потрібно використовувати функцію super()?
- Що таке декоратор?
- Які декоратори ви знаєте?
- Що таке UML-діаграми?
- Як створити діаграму класів у IDE?

Додаткові завдання

Завдання 1
Створіть ієрархію класів транспортних засобів. У загальному класі опишіть загальні для всіх транспортних засобів поля, у спадкоємцях – специфічні їм. Створіть кілька екземплярів. Виведіть інформацію щодо кожного транспортного засобу.

Завдання 2
Створіть UML-діаграму для додаткового завдання 1. Збережіть її у форматі *.uml.

Самостійна діяльність учня

Завдання 1
Створіть клас Editor, який містить методи view_document та edit_document. Нехай метод edit_document виводить на екран інформацію про те, що редагування документів недоступне для безоплатної версії. Створіть підклас ProEditor, у якому цей метод буде перевизначено. Введіть ліцензійний ключ із

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

клавіатури та, якщо він коректний, створіть екземпляр класу ProEditor, інакше Editor. Викличте методи перегляду та редагування документів.

Завдання 2
Опишіть класи графічного об'єкта, прямокутника й об'єкта, який може обробляти натискання мишки. Опишіть клас кнопки. Створіть об'єкт кнопки та звичайного прямокутника. Викличте метод натискання на кнопку.

Завдання 3
Створіть ієрархію класів із використанням множинного успадкування. Виведіть на екран порядок вирішення методів для кожного класу. Поясніть, чому лінеаризація цих класів виглядає саме так.

Завдання 4
Створіть UML-діаграми до завдань 1 та 3. Збережіть їх у форматі *.uml.

Завдання 5
Використовуючи код example_10, створіть декоратор @staticmethod для визначення повноліття людини в Україні й Америці.

Завдання 6
Використовуючи код example_10, створіть декоратори @classmethod для формування переліку об'єктів, які підрахують кількість повнолітніх людей в Україні й Америці.

Рекомендовані ресурси

Документація Python
https://docs.python.org/3/tutorial/classes.html#inheritance
https://docs.python.org/3/tutorial/classes.html#multiple-inheritance
https://www.python.org/download/releases/2.3/mro/

Статті у Вікіпедії про ключові поняття, розглянуті на цьому уроці
https://ru.wikipedia.org/wiki/Наследование_(программирование)
https://ru.wikipedia.org/wiki/Множественное_наследование
https://ru.wikipedia.org/wiki/C3-линеаризация
https://ru.wikipedia.org/wiki/Диаграмма_классов
https://ru.wikipedia.org/wiki/Декоратор_(шаблон_проектирования)
https://ru.wikipedia.org/wiki/Абстракция_данных

Стаття про порядок вирішення методів у Python
http://habrahabr.ru/post/62203/

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