

Python Essential

Наслідування

Python Essential

План заняття

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

Python Essential

Після уроку обов'язково



Повторіть цей урок у форматі відео на [ITVDN.com](http://itvdn.com)

Доступ можна отримати через керівництво вашого навчального центру



Перевірте як Ви засвоїли цей матеріал на [TestProvider.com](http://testprovider.com)

Python Essential

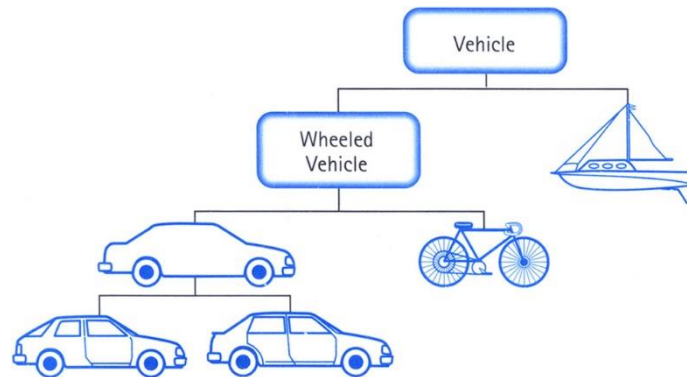
Тема

Наслідування

Наслідування

Наслідування

- **Наслідування** — механізм мови, що дозволяє описати новий клас на основі вже існуючого (батьківського, базового) класу.
- Клас-нащадок може додати власні методи та властивості, а також користуватися батьківськими методами та властивостями.
- Дозволяє будувати ієрархії класів.
- Є одним із основних принципів об'єктно-орієнтованого програмування.



Наслідування та поліморфізм

Класи старого та нового типу

- У версіях до 2.2 деякі об'єктно орієнтовані можливості Python були помітно обмежені. Починаючи з версії 2.2, об'єктна система Python була суттєво перероблена та доповнена.
- Для сумісності з існуючим кодом у Python 2 існують дві системи типів: *класи нового типу* (new-style classes) і *класи старого типу* (old-style classes, classic classes).
- Для створення класу нового типу слід успадкувати його від іншого класу нового типу. Усі стандартні класи є класами нового типу. Базовим є клас **object**.
- Якщо в Python 2 не вказувати базовий клас або успадкувати його від іншого класу старого типу, цей клас є класом старого типу.



У Python 3 всі класи є класами нового типу і успадковуються за замовчуванням **object**.



Класи старого типу потрібні лише зворотної сумісності. У новому кодi слід використовувати лише класи нового типу.

Наслідування

Реалізація наслідування

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

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

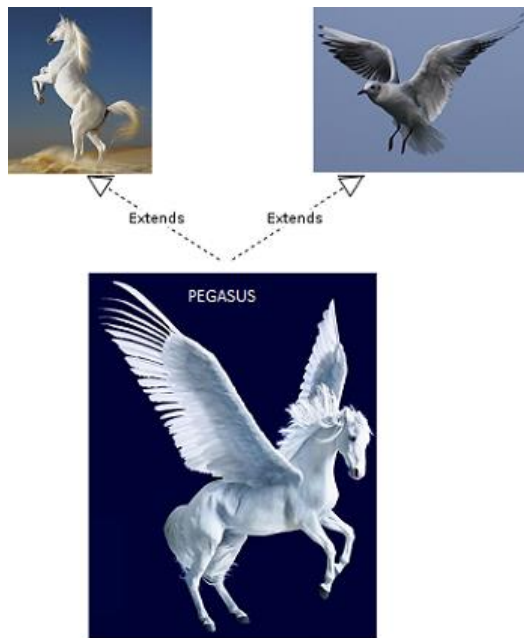
class UpdateCat(Cat):
    def bite_a_finger(self):
        print("Bite! :)")

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

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

Наслідування

Множинне наслідування



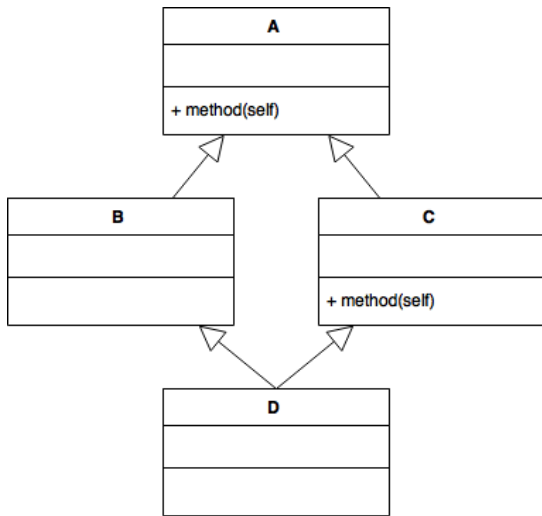
При **множинному наслідуванні** у класа може бути більше одного класа-батька. І тут клас успадковує методи всіх класів-батьків. Гідність такого підходу є більшою гнучкістю, проте він може бути потенційним джерелом помилок.

Список базових класів вказується через кому в круглих дужках після імені цього класу:

```
class Pegasus(Horse, Bird):  
    pass
```


Наслідування

Порядок дозволу методів (MRO)



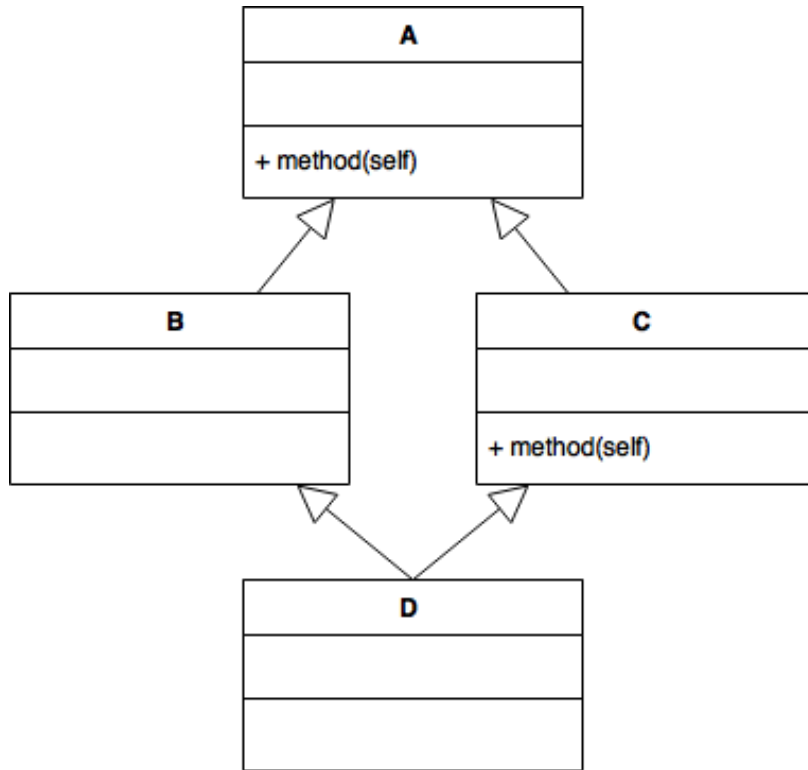
- Якщо атрибут, до якого здійснюється доступ, не знайдений у поточному класі, здійснюється його пошук у класах-батьках.
- Порядок, в якому інтерпретатор продивляється базові класи, визначається **лінеаризацією** даного класу, яка також називається **MRO (Method Resolution Order)**. Вона зберігається в атрибуті класу `__mro__`.
- MRO будується за допомогою алгоритму C3-лінеаризації.
- Властивості лінеаризації:
 - *стійкість та розширюваність*;
 - *монотонність*: у лінеаризації класу-нащадка дотримується той самий порядок прямування класів-батьків, що й у лінеаризації класу-батька;
 - *властивість локального старшинства*: в лінеаризації класу-нащадка дотримується той самий порядок слідування класів-батьків, що у його оголошенні



Лінеаризація будується лише для класів нового типу. У класах старого типу пошук атрибутів провадиться за допомогою пошуку в глибину, що може давати некоректні результати при ромбоподібному наслідуванні.

Наслідування

Порядок дозволу методів (MRO)



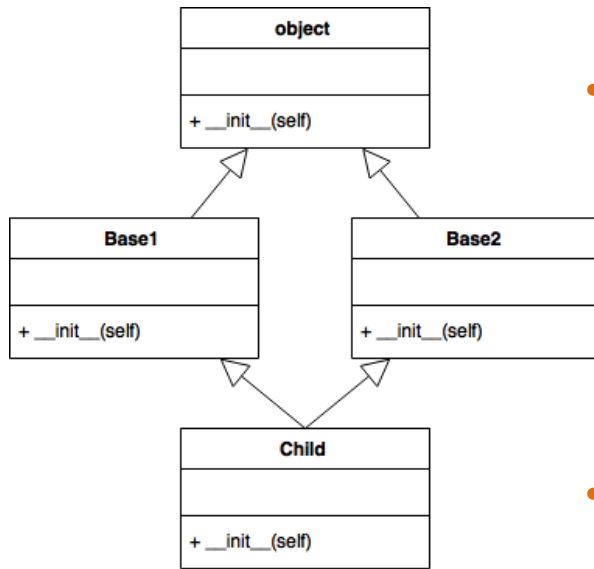
У прикладі ліворуч лінеаризація класу D, тобто `D.__mro__` виглядає як `[D, B, C, A]`.

Таким чином, якщо спробувати викликати `obj.method()`, де `obj` – екземпляр класу D, буде викликано метод класу C.

Якби A був класом старого типу в Python 2, то був би викликаний метод `method` класу A, що в даному випадку є неправильною поведінкою, оскільки він перевизначений у класі C, спадкоємцем якого є клас D. Це одна з причин, за якими слідє Використовувати класи нового типу.

Наслідування

Отримання доступу до атрибутів суперкласу



- Якщо в даному класі метод або атрибут був перевизначений, а потрібен доступ до відповідного атрибуту суперкласу, це можна зробити двома способами:
- шляхом явного звернення до атрибуту необхідного класу:
`BaseClass.method(self)`
 - за допомогою інстанціювання спеціального проксі-об'єкта класу `super` (виглядає, як виклик функції).
 - У Python 2 як параметри конструктора `super` передаються ім'я поточного класу та посилання на екземпляр поточного класу:
`super(MyClass, self).method()`
- В Python 3 можна не вказувати нічого і дані параметри будуть отримані автоматично:
`super() .method()` # те саме, що `super(__class__, self).method()`



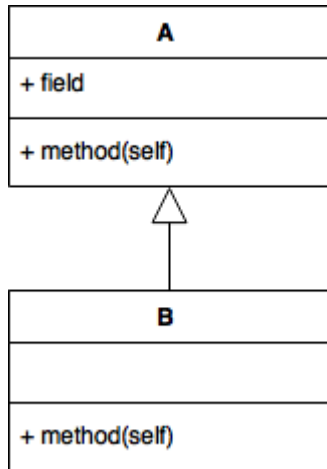
super недоступний для класів старого типу



super повертає спеціальний проміжний об'єкт, який надає доступ до атрибутів наступного класу `__mro__`.

Наслідування

Визначення типу об'єкта



- Тип* цього об'єкта можна визначити за допомогою атрибуту `__class__` та вбудованої функції `type(obj)`.
- Атрибут класу `__bases__` зберігає кортеж (незмінний список) базових класів.
- Оскільки ставлення успадкування є транзитивним, у випадку для перевірки того, чи є даний об'єкт екземпляром заданого класу чи є даний клас підкласом заданого класу, ці атрибути потрібно перевіряти рекурсивно. Існують вбудовані функції, які це роблять.
- `isinstance(obj, cls)` перевіряє, чи є `obj` екземпляром класу `cls` або класу, який є спадкоємцем класу `cls`;
- `issubclass(cls, base)` перевіряє, чи є клас `cls` спадкоємцем класу `base`.



* Примітка: оскільки в Python все є об'єкт і відсутні примітивні типи даних, зазвичай терміни «тип» і «клас» є синонімами.

Наслідування

Декоратор @staticmethod

```
class MyClass:  
    def __init__(self, name):  
        self.name = name
```

```
    def getName(self):  
        return self.name
```

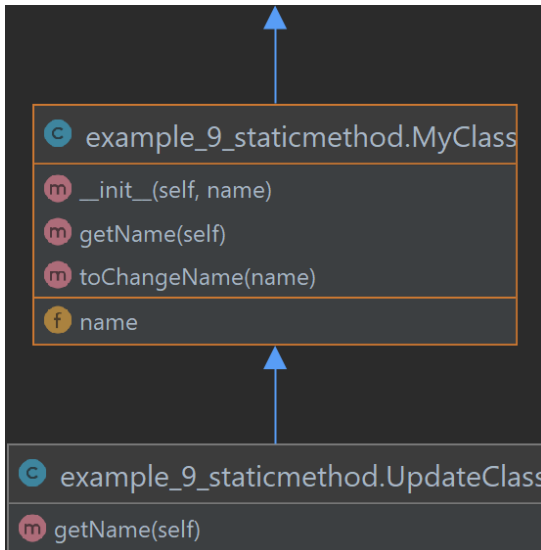
@staticmethod

```
    def toChangeName(name):  
        return name.replace("/", "-")
```

```
class UpdateClass(MyClass):  
    def getName(self):  
        return MyClass.toChangeName(self.name)
```

```
name1 = MyClass("25-07-2022")  
ud_name1 = UpdateClass("25/07/2022")
```

```
if name1.getName() == ud_name1.getName():  
    print("Equal")  
else:  
    print("Unequal")
```



Наслідування

Декоратор @classmethod

```
from datetime import date
```

```
class MyClass1:
```

```
    def __init__(self, surname, name, age):
        self.name = name
        self.name = name
        self.age = age
```

```
@classmethod
```

```
def fromBirthYear(cls, surname, name, birthYear):
    return cls(surname, name, date.today().year - birthYear)
```

```
def display(self):
    print(self.name + "'s age is: " + str(self.age))
```

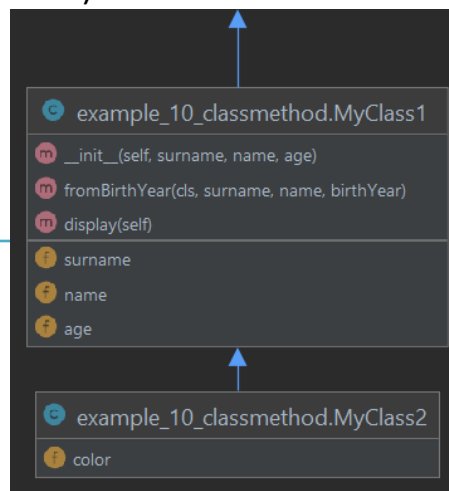
```
class MyClass2(MyClass1):
    color = 'White'
```

```
m_per1 = MyClass1('Ivanenko', 'Ivan', 19)
m_per1.display()
```

```
m_per2 = MyClass1.fromBirthYear('Dovzhenko', 'Bogdan', 2000)
m_per2.display()
```

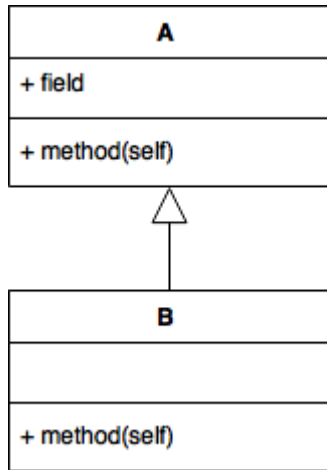
```
m_per3 = MyClass2.fromBirthYear('Sydorchuk', 'Petro', 2010)
print(isinstance(m_per3, MyClass2)) # True
```

```
m_per4 = MyClass2.fromBirthYear('Makuschenko', 'Dmytro', 2001)
print(isinstance(m_per4, MyClass1)) # True
print(issubclass(MyClass1, MyClass2)) # False
print(issubclass(MyClass2, MyClass1)) # True
```

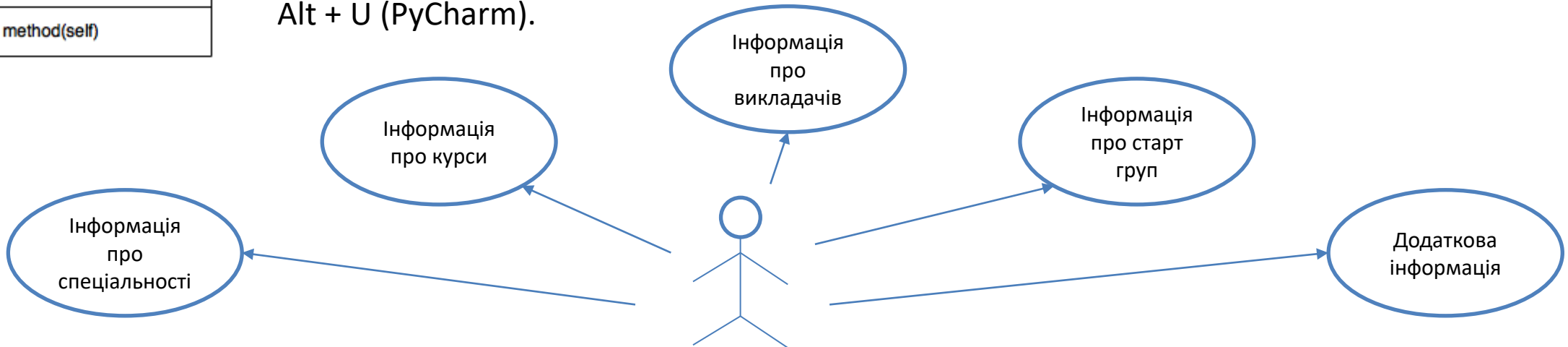


Наслідування

Створення діаграми класів



- **Unified Modeling Language (UML)** – уніфікована мова моделювання.
- Modeling має на увазі створення моделі, що описує об'єкт.
- Unified (універсальний, єдиний) — підходить для широкого класу програмних систем, що проектується, різних областей додатків, типів організацій, рівнів компетентності, розмірів проектів.
- 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.
- Для створення діаграми класів необхідно використати комбінацію клавіш: Ctrl + Shift + Alt + U (PyCharm).



Дивіться наші уроки у відео форматі

ITVDN.com



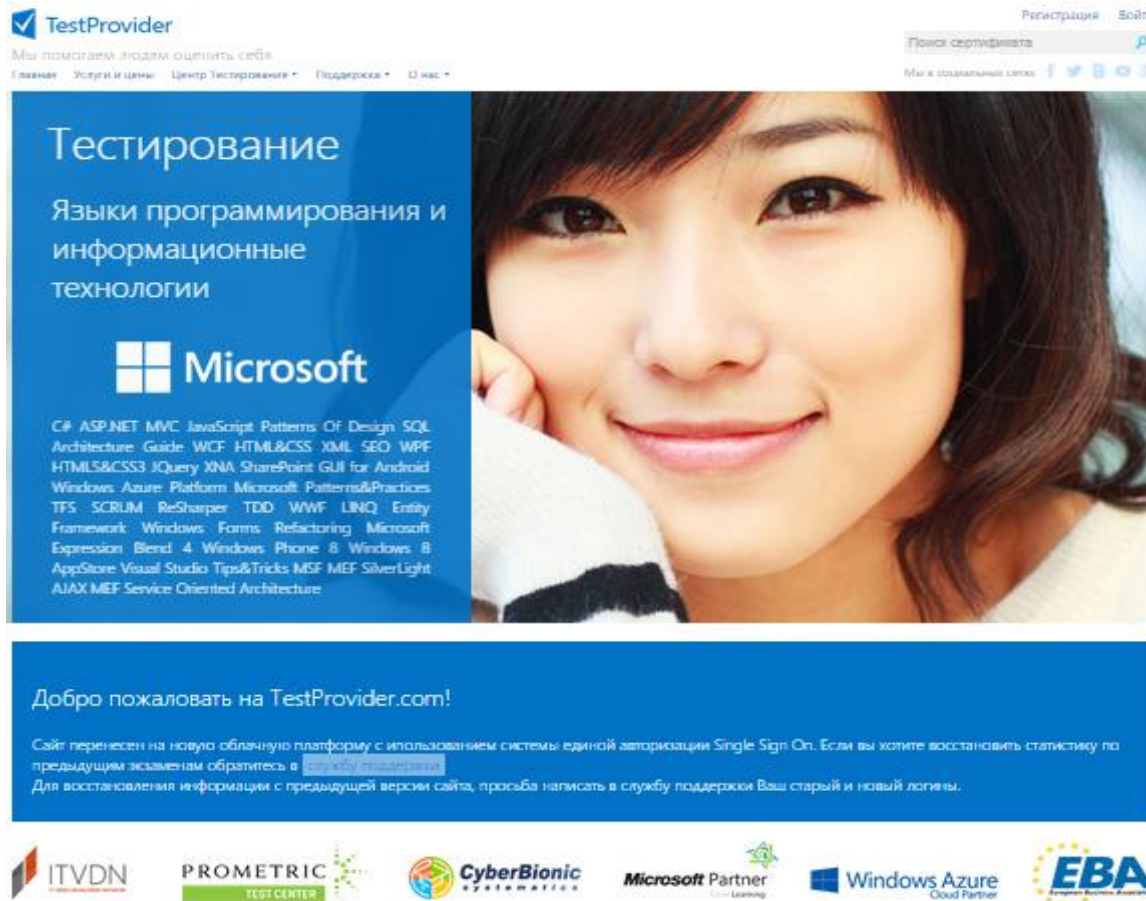
Перегляньте цей урок у відео форматі на освітньому порталі [ITVDN.com](http://itvdn.com) для закріплення пройденого матеріалу.

Усі курси записані сертифікованими тренерами, які працюють у навчальному центрі CyberBionic Systematics.



Перевірка знань

TestProvider.com



TestProvider – це online сервіс перевірки знань з інформаційних технологій. За його допомогою Ви можете оцінити Ваш рівень та виявити слабкі місця. Він буде корисним як у процесі вивчення технології, так і загальної оцінки знань IT фахівця.

Після кожного уроку проходите тестування для перевірки знань на TestProvider.com

Успішне проходження фінального тестування дозволить Вам отримати відповідний Сертифікат.



Python Essential

Q&A

Інформаційний відеосервіс для розробників програмного забезпечення

