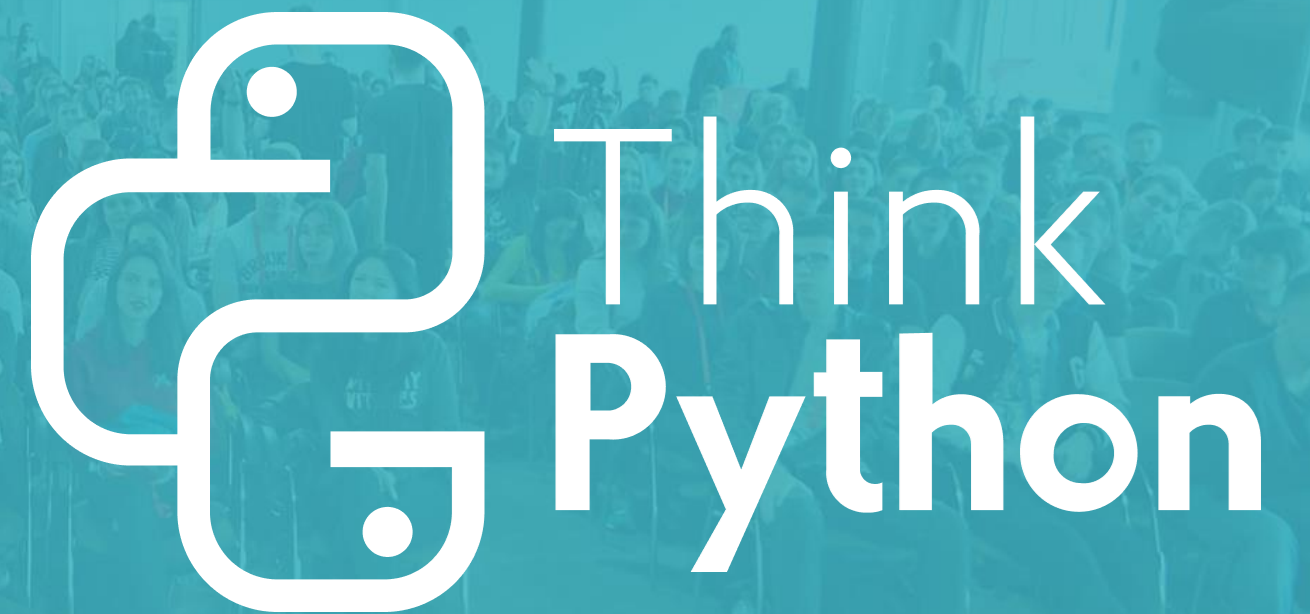




Протоколи в Python

керована еволюція та стандартизація
взаємодії між об'єктами

Світлана Сумець



▶ ЩО СЬОГОДНІ РОЗГЛЯНЕМО?

01

Які бувають види типізації?

02

Яка типізація в Python?

03

Яку типізацію реалізують протоколи?

04

Що таке протоколи?

05

Як та навіщо їх використовувати?

06

У чому різниця протоколів та абстрактних класів?



ТИПІЗАЦІЯ

► КЛАСИФІКАЦІЇ ТИПІЗАЦІЇ

Як типи перевіряються
під час виконання програми

- статична типізація
- динамічна типізація

Як типи визначаються
і порівнюються між собою

- номінальна типізація
- структурна типізація

▶ СТАТИЧНА ТА ДИНАМІЧНА ТИПІЗАЦІЇ

ДИНАМІЧНА

- ▶ Типи даних визначаються під час виконання програми
- ▶ Змінні можуть містити значення різних типів у різні моменти часу

СТАТИЧНА

- ▶ Типи даних визначаються на етапі компіляції програми і не можуть змінюватися протягом її виконання
- ▶ Змінні мають фіксований тип, який визначається в явному порядку при оголошенні змінної

► КЛАСИФІКАЦІЇ ТИПІЗАЦІЇ

Як типи перевіряються
під час виконання програми



СТАТИЧНА ТИПІЗАЦІЯ



ДИНАМІЧНА ТИПІЗАЦІЯ

Як типи визначаються
і порівнюються між собою



НОМІНАЛЬНА ТИПІЗАЦІЯ



СТРУКТУРНА ТИПІЗАЦІЯ

▶ НОМІНАЛЬНА ТА СТРУКТУРНА ТИПІЗАЦІЇ

НОМІНАЛЬНА

▶ Типи збігаються, якщо вони мають однакові імена, навіть якщо їх структури відрізняються

▶ **Немає у Python!**

Тайпхінти лише визначають семантику номінальної типізації

СТРУКТУРНА

▶ Типи збігаються, якщо вони мають однакові структури або форми даних

▶ **Немає у Python!**

Качина типізація лише визначає семантику структурної типізації

Якщо щось **виглядає**, як
качка, **плаває**, як качка, і
крякає, як качка, то це,
ймовірно, і є **качка**



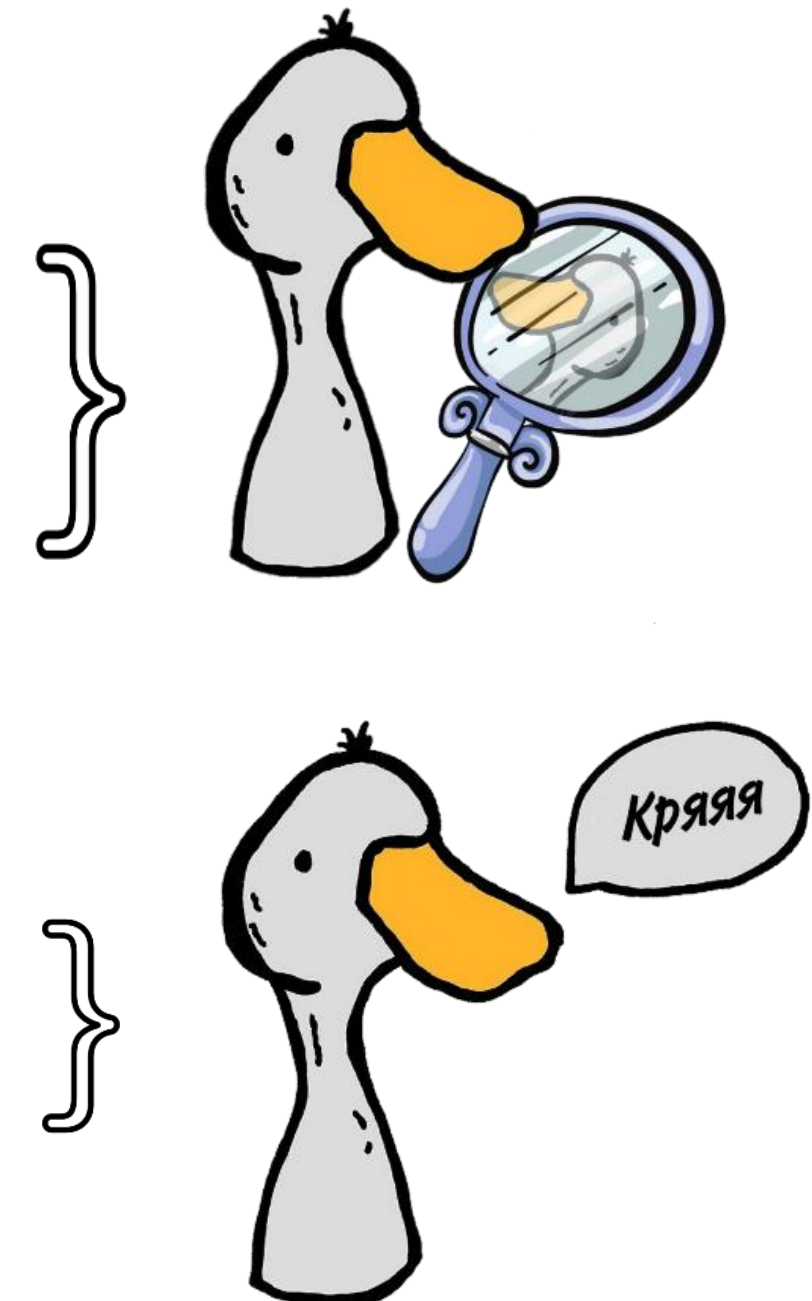
► КАЧИНА ТИПІЗАЦІЯ



```
class Duck:
    def __init__(self, beak, wings):
        self.beak = beak
        self.wings = wings

    def swim(self):
        return 'I swim'

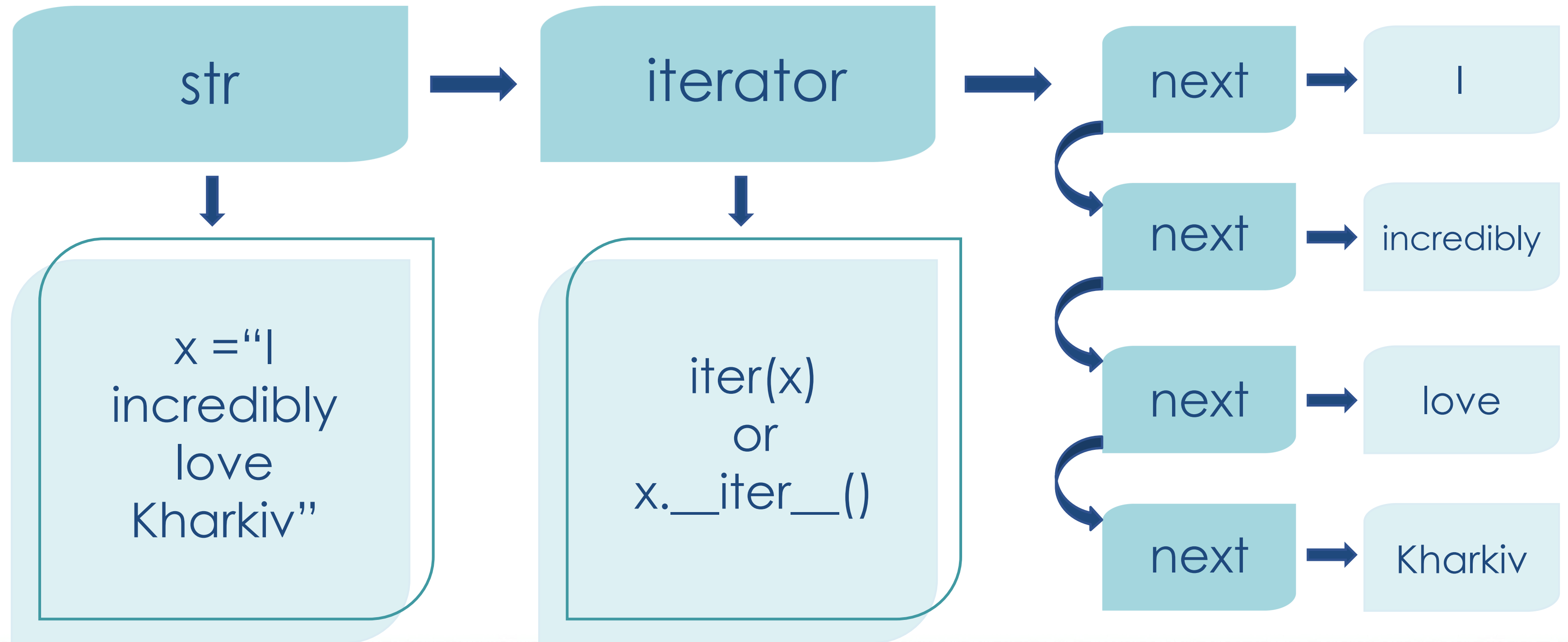
    def quack(self):
        return 'I quack'
```





ПРОТОКОЛИ

► ПРОТОКОЛ ИТЕРАТОРА




► ПРОТОКОЛИ

Протоколи є альтернативою абстрактним базовим класам, вони дають змогу користуватися структурною типізацією, тобто здійснювати перевірку сумісності класів виключно на основі аналізу їхніх атрибутів і методів

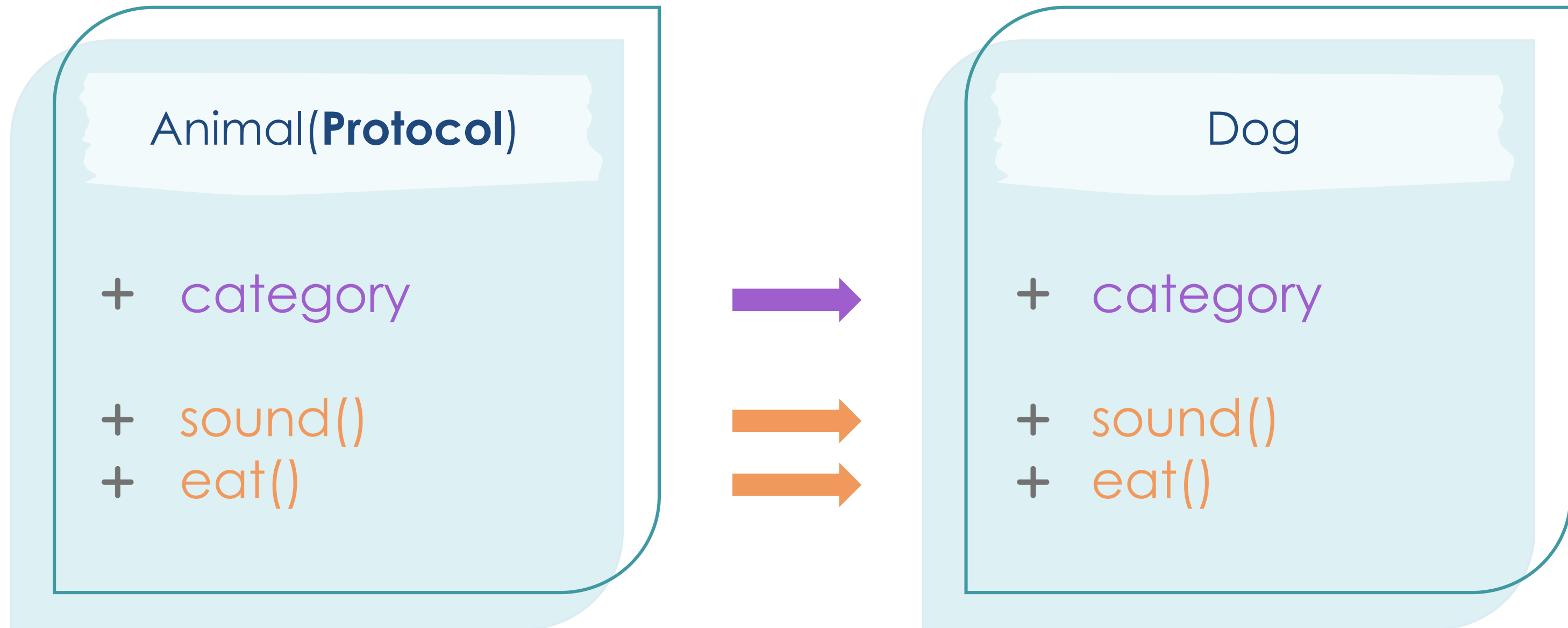


Чому протокол
завжди був
такий
популярний?..



Бо в нього є
потрібні методи,
а інші ними вміло
користуються

▶ ПРОТОКОЛИ





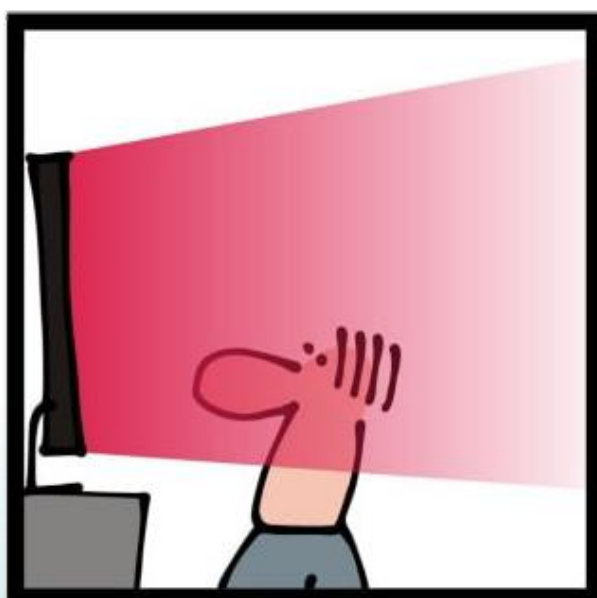
Створю
протокол *Animal*
та клас *Dog*, що
його реалізує

► @RUNTIME_CHECKABLE



```
isinstance(Dog, Animal)  
issubclass(Dog, Animal)
```

} перевірка
відповідності типів у
момент виконання
програми



TypeError: Instance and class
checks can only be used with
@runtime_checkable protocols

► ІЄРАРХІЯ ПРОТОКОЛІВ



01

Клас-протокол має успадковуватися від `typing.Protocol`

02

Ієрархії протоколів повинні складатися виключно з протоколів

03

Є обмеження на розширення непротокольних класів



АБСТРАКТНІ БАЗОВІ КЛАСИ

▶ АБСТРАКТНІ КЛАСИ

Абстрактні базові класи є способом визначення загального інтерфейсу для підкласів, що вимагає реалізації певних методів.

Вони надають шаблон для створення ієрархії класів і забезпечують контроль над тим, які методи мають бути реалізовані у підкласах..



► ПЕРЕВАГИ АБСТРАКТНИХ КЛАСІВ

АВС сприяють повторному використанню коду

АВС ідеальні для управління ієрархією класів

Протоколи не реалізують загальну функціональність

Протоколи не покладаються на жорстку ієрархію



► ПЕРЕВАГИ ПРОТОКОЛІВ

Явне спадкування для
сумісності з **ABC**

ABC для підконтрольного
проектом коду

Протоколи не вимагають
змін в ієрархії

Протоколи як інтерфейси
для сторонніх бібліотек



► ОТЖЕ

ПРОТОКОЛИ

АБСТРАКТНІ КЛАСИ

Гнучкість

Висока гнучкість, дозволяють реалізацію заднім числом і сумісність між типами

Структурований підхід для дотримання певного інтерфейсу в ієрархії класів

Зворотна сумісність

Реалізація необхідних методів та атрибутів заднім числом без змін в ієрархії класів

Явне успадкування, гарний вибір для інтерфейсів з нуля та контролю над ієрархією

Сторонні бібліотеки

Корисні для визначення інтерфейсів для сторонніх бібліотек без зміни їх коду

Не підтримують підходи без чіткої ієрархії класів, працюють з підконтрольним кодом

▶ ДЖЕРЕЛА ПОСИЛАННЯ

КОРИСНО ПОЧИТАТИ

- ▶ [PEP 544 – Protocols: Structural subtyping](#)
- ▶ [Static Duck Typing With Python's Protocols](#)
- ▶ [Abstract Base Classes and Protocols: What Are They? When To Use Them?](#)
- ▶ [Python Protocols vs. ABCs: Choosing The Right Approach for Interface Design](#)

nix

ДЯКУЮ ЗА УВАГУ!

