ОСНОВИ СИСТЕМ ШТУЧНОГО ІНТЕЛЕКТУ, НЕЙРОННИХ МЕРЕЖ ТА ГЛИБОКОГО НАВЧАННЯ

Модуль 6. ВИСОКОРІВНЕВА МОВА ПРОГРАМУВАННЯ PYTHON

Лекція 6.6. Об’єктно-орієнтоване програмування. Проектування класів.

КЛАСИ

Класи Python реалізують усі основні концепції ООП:

- **Інкапсуляція**: об'єднання даних та методів
- **Спадкування:** створення нового класу на базі існуючого
- **Поліморфізм:** використання об'єктів з однаковим інтерфейсом без інформації про тип і внутрішню структуру об'єкта

СТВОРЕННЯ КЛАСУ

class <name> (superclass, …):
<тіло класу>

Кожна інструкція **class** створює новий об'єкт типу **клас.**

Кожний раз, коли викликається клас, він створює новий об'єкт – **екземпляр** класу.

Екземпляри автоматично зв'язуються с класом із яких вони були створені.

Класи зв'язані із своїми суперкласами, які вказані в круглих дужках в інструкції **class.** Послідовність суперкласів у списку визначає порядок розташування в дереві пошуку імен.

СТВОРЕННЯ КЛАСУ

class <name> (superclass, …):
<тіло класу>

class – інструкція створення об'єкту, посилання на який зберігається в просторі імен як **<ім'я класу>.**

<тіло_класу> сукупність інструкцій. Може містити присвоювання змінним **data=value** (**data** стає **атрибутом - полем** класу) та визначення функцій **def …(self, …): ….** (стає **атрибутом - методом** класу).

Першим аргументом методу завжди є екземпляр класу, для якого викликається цей метод. За угодою, цей аргумент називається **"self"**. Спеціальний метод **__init__( )** – конструктор – викликається автоматично при створенні екземпляра класу.

СТВОРЕННЯ КЛАСУ

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

    def prntvl (self)
        print('value =', self.value)

Myobj = MyClass(3.14)
Myobj.prntvl()
Myobj.value = "НОНОНО"
Myobj.prntvl()
```

СТВОРЮЄ об'єкт типу **КЛАС**

Конструктор

Функція класу

СТВОРЮЄ об'єкт типу **ЕКЗЕМПЛЯР КЛАСУ**

Звернення до аргументу (методу)

→ *3.14*

Звернення до аргументу (поле)

→ *НОНОНО*

ПРОСТІР ІМЕН

Виклик об'єкта класу як функції створює новий об'єкт екземпляру.

Кожен об'єкт екземпляру успадковує атрибути класу і набуває **свій власний простір імен.** Вони спочатку **порожні**, але успадковують атрибути класів, з яких були створені.

Методи класу отримують в першому аргументі (з ім'ям ***self***) посилання на оброблюваний об'єкт – екземпляр. Присвоювання атрибутів через посилання ***self*** створює чи змінює дані **екземпляру**, а не класу.

ПРОСТІР ІМЕН

```
class Cls_Empty :
    pass
```

Список імен поточної області видимості

dir() → *['In', 'Out' , ..., Cls_Empty',...]*

dir(Cls_Empty)

Список атрибутів класу

→ *['__class__', '__doc__' , '__dict__',.....]*

Cls_empty.__dict__

Словник класу

→ *({'__module__': '__main__' ,*
'__doc__' : None',
__dict__': <...>, ... })

Cls_empty.X = 25; Cls_empty.Y = 42
Cls_empty.__dict__

→ *({..., 'X' : 25, 'Y': 42})*

Словник класу
!!! X Y

ПРОСТІР ІМЕН

ex_1 = Cls_empty()
ex_2 = Cls_empty()

2 екземпляри класу

ex_1.X → *25*

ex_2.Y → *42*

ex_1.X = ‘HOHOHO’
ex_2.Y = ‘146823’

Змінюються атрибути екземплярів, но не класу

ex_1.X → *‘HOHOHO’*

ex_1.Y → *42*

ex_2.X → *25*

ex_2.Y → *146823*

cl_empty.X → *25*

cl_empty.Y → *42*

ПРОСТІР ІМЕН

```
class MixedData :
    data = 'STRING'
    def __init__(self, value):
        self.data = value

def dispout(self):
        print (self.data, MixedData.data)

x = MixedData ('FIRST')
y = MixedData (2)

x.dispout()
y.dispout()
```

data - атрибут класу

data - атрибут екземпляру

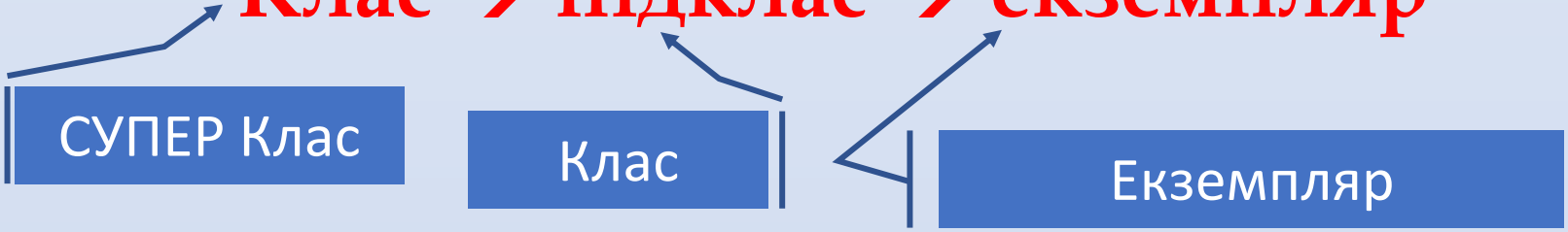
Функція друку атрибуту класу та атрибуту екземпляру

→ *FIRST STRING*

→ *2 STRING*

СПАДКУВАННЯ: СУПЕРКЛАС, ПІДКЛАС

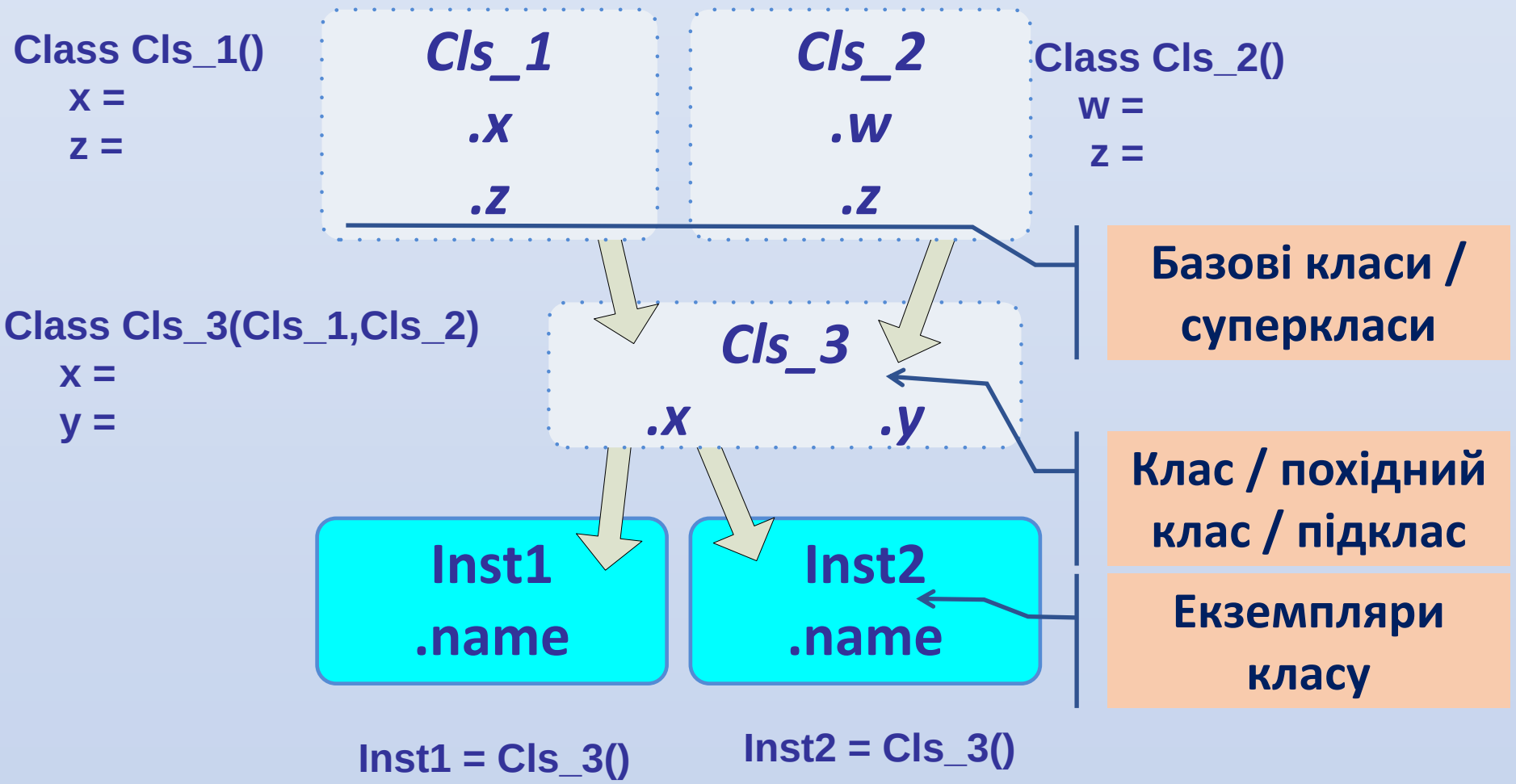
Клас → підклас → екземпляр



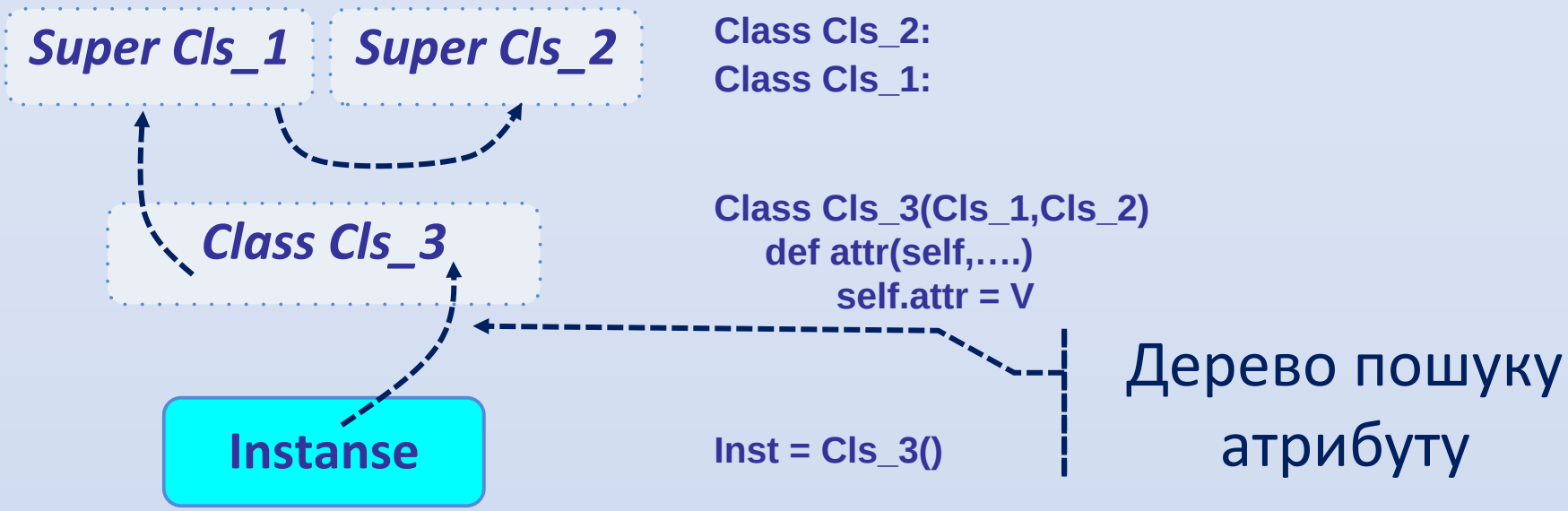
Суперкласи перераховуються в круглих дужках в заголовку інструкції ***class***. Клас, що наслідує, називається **підкласом**, а клас, який унаслідується, називається його **супер-класом**.

Клас наслідує **атрибути суперкласу, екземпляр** наслідує **атрибути класу.**

ДЕРЕВО КЛАСІВ: СУПЕРКЛАС, КЛАС, ЕКЗЕМПЛЯР



ДЕРЕВО УСПАДКУВАННЯ



inst.attr ?

Атрибути екземплярів створюються за допомогою присвоювання атрибутів аргументу ***self*** в методах класу. Атрибути класів створюються інструкціями (привласнення), розташованої всередині інструкції **class**.

Посилання на суперклас - перерахування класів в круглих дужках в заголовку інструкції **class**.

СИНТАКСИС КВАЛІФІКАЦІЇ ІМЕН

Для доступу до атрибутів будь-якого об'єкта використовується синтаксис кваліфікації імені ***object.attribute*** **->** це вираз, що повертає значення, яке присвоєно імені атрибуту.

Звернення до імен, **кваліфікуючи їх**, явно вказує інтерпретатору об'єкт, атрибут якого потрібно використати.

- Прості змінні (правило **LEGB**), наприклад **X** - пошук імені **X** в поточних областях видимості.
- Кваліфіковані імена **X.Y** - пошук імені **X** спочатку в поточних областях видимості, а потім буде виконаний пошук атрибута **Y** в об'єкті **X** (не в областях видимості).
- Кваліфіковані шляхи **X.Y.Z** - буде проведений пошук імені **Y** в об'єкті **X**, а потім пошук імені **Z** в об'єкті **X.Y**.

СИНТАКСИС КВАЛІФІКАЦІЇ ІМЕН

додаток для класів

Пошук в дереві успадкування кваліфікованих імен типу ***object.X***

Посилання *object.X* → пошук атрибута **X** виконується спочатку в об'єкті ***object***, потім у всіх класах, розташованих вище в дереві спадкоємства. 14

Присвоєння *object.X = value* → створюється чи змінюється атрибут з ім'ям **X** в просторі імен об'єкта ***object*, і нічого більше**.

ВБУДОВАНІ АРГУМЕНТИ КЛАСУ

Атрибут	
__name__	Ім'я класу
__module__	Ім’я модуля
__dict__	Словник атрибутів класу
_bases__	Кортеж суперкласів
__doc__	Рядок документації класу

ВБУДОВАНІ АРГУМЕНТИ ЕКЗЕМПЛЯРУ

Атрибут	
__dict__	**Словник атрибутів класу**
__class__	**Клас, що є «батьком» екземпляру**
__init__	**Конструктор**
__del__	Деструктор (фіналізатор)
__cmp__	Викликається для порівняння (нема в версіях >=3.0)
__hash__	Хеш-значення обєкту
__getattr__	Вертає значення атрибуту
__setattr__	Встановлює значення атрибуту
__delattr__	Видаляє атрибут
__call__	Виконується при виклику екземпляру

ВБУДОВАНІ АРГУМЕНТИ (приклад: __class__, __bases__)

```
def clstree (cls, indent) :
    print ('.' * indent  + cls.__name__)
    for supercls  in cls.__bases__ :     #!рекурсія суперкласів
        clstree(supercls, indent+4)
def instancetree (inst) :
    print ('tree of ', inst)
    clsstree (inst.__class__, 4)

class A              : pass
class B(A)           : pass
class C(A)           : pass
class D(B,C)         : pass
class E              : pass
class F(D,E)         : pass

instancetree (B())                  # що буде надруковано?
```

Рекомендована ЛІТЕРАТУРА

- **Програмування числових методів мовою Python:** підруч. / А. В. Анісімов, А. Ю. Дорошенко, С. Д. Погорілий, Я. Ю. Дорогий ; за ред. А. В. Анісімова. – К. : Видавничо-поліграфічний центр "Київський університет", 2014. – 640 с.
- **Програмування числових методів мовою Python:** навч. посіб. / А. Ю. Дорошенко, С. Д. Погорілий, Я. Ю. Дорогий, Є. В. Глушко ; за ред. А. В. Анісімова. – К. : Видавничо-поліграфічний центр "Київський університет", 2013. – 463 с.
- **Основи програмування Python:** Підручник для студ. спеціальності 122 «Компютерні науки» / А.В.Яковенко; КПІ.- Київ: КПІ, 2018 . – 195 с.
- **Лутц М.** Изучаем Python, 4-е издание. - СПб.: Символ-Плюс. 2011.- 1280 с.: ил.

Рекомендовані посилання

- **Python OOP Tutorials – Working with Classes: 6 lessons.** – https://www.youtube.com/watch?v=ZDa-Z5JzLYM&list=PL-osiE80TeTsqhIuOqKhwlXsIBIdSeYtc

Контрольні запитання

- Обґрунтуйте необхідність та вкажіть переваги використання класів.
- Поясніть відмінності класу і екземпляру класу. Надайте приклад створення класу и екземпляру.
- Поясніть призначення об'єкту **self** та надайте приклади його застосування.
- Поясніть використання аргументу **__init__** при створенні класу, надайте приклади.
- Поясніть призначення вбудованого аргументу класу **__bases__** .
- Поясніть призначення вбудованого аргументу класу **__doc__** .
- Поясніть призначення вбудованого аргументу екземпляру **__dict__** .

The END
Модуль 6. Лекція 6.6.