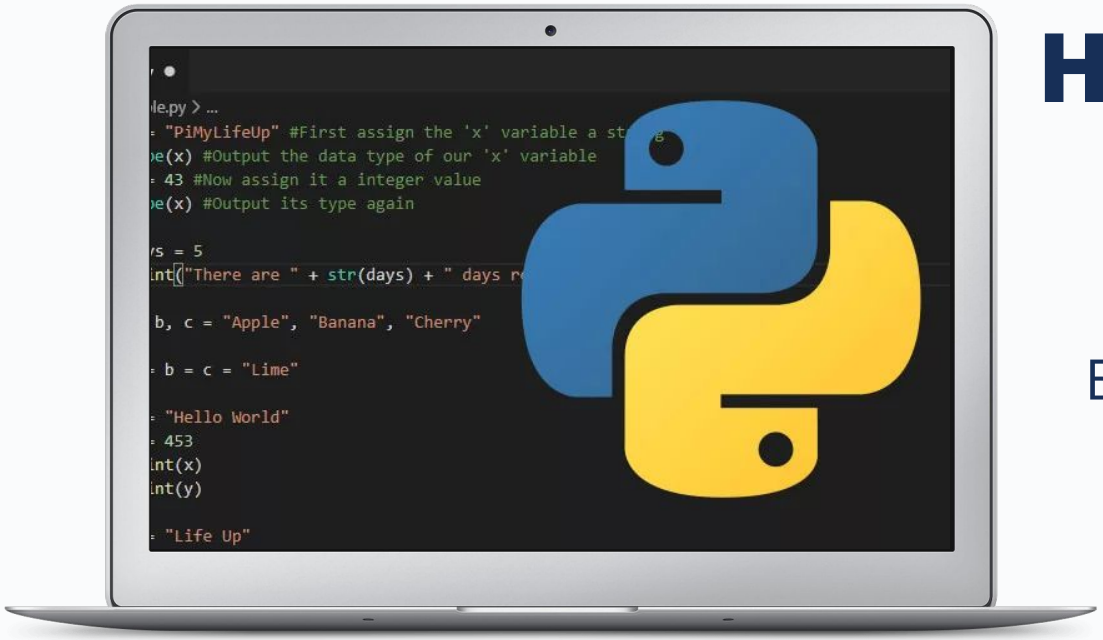
Become QA Auto

PROMETHEUS



Наслідування

Бутенко Сергій

План лекції

- Поняття наслідування
- Успадкування методів
- Успадкування атрибутів
- Множинне наслідування
- Успадкування конструкторів

Поняття наслідування

Наслідування - це властивість, що дозволяє описати новий клас на основі вже наявного з частковим чи повним збереженням функціональності.

Новий клас, що наслідує батьківський навають - **нащадком**, **підкласом** або **дочірнім** класом.

Наслідування - забезпечує повторне використання існуючого коду без його копіювання.

Наслідування реалізує відношення **“is-a”** (“є”) між суперкласом і підкласом.

Базовий, суперклас або **батьківський** - це клас, від якого наслідується новий.

Синтаксис організації наслідування

```
class <ім'я підкласу>(<ім'я базового класу>):
```

Побудова ієрархії наслідування

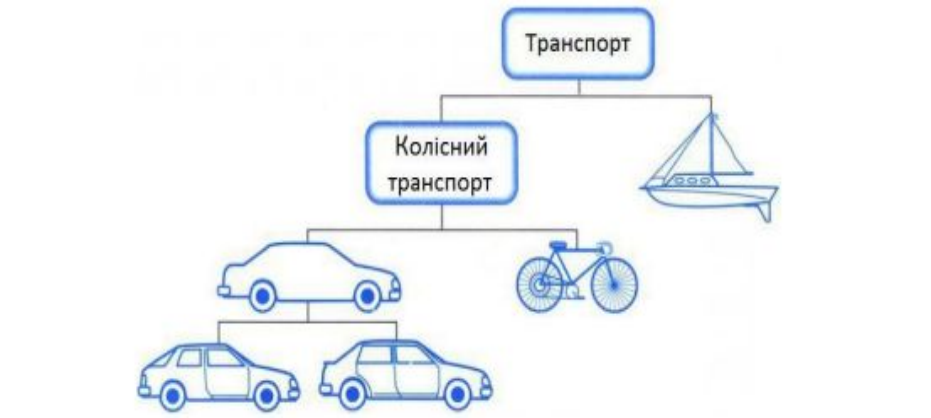
Наслідування дозволяє створювати ієрархічні структури об'єктів.

object - це базовий клас для всіх класів, бо від цього класу явно або неявно успадковуються всі інші класи

Приклад ієрархії:

```
class BaseClass:
    pass
class DerivedClass(BaseClass):
    pass
```

Вбудована функція issubclass(cls, super_class) визначає чи певний клас є підкласом іншого класу по всій ієрархії успадкування



Успадкування атрибутів

Дочірній клас **успадковує відкриті** (публічні) атрибути базового класу.

Згідно механізму **“name mangling”** приватні атрибути за межами класу мають ім'я:

`_<ім'я базового класу>__<ім'я атрибуту>`

У дочірньому класі **приватні** атрибути базового класу **не успадковуються**

Використання приватних атрибутів дозволяє "приховати" внутрішню реалізацію базового класу і для дочірніх класів.

Приклад 1. Успадкування атрибутів

shopInherit.py

PYTHON_BASICS > shopInherit.py > ...

```
1  class ShopWorker:
2      _count_workers=0
3      def __init__(self, name1='', age1=0):
4          self.name = name1
5          self.__age = age1
6          self.setting_id()
7      def setting_id(self):
8          ShopWorker._count_workers +=1
9          self.id = ShopWorker._count_workers
10     def add_year(self):
11         self.__age+=1
12     def __str__(self):
13         str_out="Працівник "+str(self.id)+": "+self.name+" "+str(self.__age)
14         str_out += " всіх працівників " + str(ShopWorker._count_workers)
15         return str_out
16     @staticmethod
17     def info():
18         print('В магазині працює: ', ShopWorker._count_workers, ' працівників')
19     @classmethod
20     def naming_shop(cls, name):
21         cls.name_shop= name
22         return cls.name_shop
23     def get_age(self):
24         return self.__age
25     def set_age(self, new_age):
26         if (new_age < 0):
27             new_age= - new_age
28         self.__age=new_age
```

Опис класу **ShopWorker**
Створення атрибуту класу **count_workers**
Конструктор класу

Виклик методу **setting_id** з конструктора
Опис методу **setting_id**
Ініціалізація полів **count_workers** та **id**

Опис методу **add_year** для збільшення віку об'єкта

Опис методу **__str__**

Статичний метод info()

Метод класу naming_shop(cls, name):

Гетер для атрибуту **__age**

Сетер для атрибуту **__age**
Перевірка параметра **new_age** на допустиме значення, та його коригування

Приклад 1. Успадкування атрибутів

```python
class Seller (ShopWorker):
    pass

ShopWorker.info()

worker_one = ShopWorker('Іван', 25)
print (worker_one)
worker_one.add_year()
print (worker_one)
ShopWorker.naming_shop('Fara')
print ('Назва магазину: ',ShopWorker.name_shop)

seller1= Seller()
print(seller1)
print(seller1.name)
seller2= Seller('Марія',40)
print(seller2)
print(seller2.name)
print(Seller.name_shop)
print(Seller._count_workers)
```

TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE JUPYTER Python

```
В магазині працює:  0  працівників
Працівник 1: Іван 25 всіх працівників 1
Працівник 1: Іван 26 всіх працівників 1
Назва магазину:  Fara
Працівник 2:  0 всіх працівників 2

Працівник 3: Марія 40 всіх працівників 3
Марія
Fara
3
```

Опис класу **`Seller`**, що наслідує **`ShopWorker`**

Вивід атрибуту класу **`ShopWorker`**

Створення об'єкту **`worker_one`**
Вивід даних про працівника **`worker_one`**
Виклик методу для збільшення віку **`add_year()`**
Повторний вивід даних про працівника **`worker_one`**
Виклик методу класу **`naming_shop`** через назву класу
Вивід атрибута класу **`name_shop`**

Створення об'єкту **`seller1`**
Вивід даних про працівника **`seller1`**
Вивід поля **`name`** для працівника **`seller1`**
Створення об'єкту **`seller2`**
Вивід даних про працівника **`seller2`**
Вивід поля **`name`** для працівника **`seller2`**
Вивід атрибута класу **`name_shop`** через назву класу **`Seller`**
Вивід атрибута класу **`_count_workers`** через назву класу **`Seller`**

Результат роботи коду

Успадкування конструкторів

Дочірний клас успадковує конструктор від батьківського класу.

Розглянемо найпростішу конструкцію класу:

```
class ShopWorker:
    pass
```

Клас **ShopWorker** містить конструктор **__init__(self)** успадкований від класу **object.**

Дочірний клас при цьому може перевизначити конструктор батьківського класу (створити свій на основі батьківського).

Приклад перевизначеного конструктора в класі **ShopWorker**:

```
def __init__(self, name1='', age1=0):
        self.name = name1
        self.__age = age1
        self.setting_id()
```

Клас **Seller** успадковує від класу **ShopWorker** цей конструктор

Успадкування конструкторів

При **перевизначенні** конструктора першою інструкцією заміщеного конструктора має бути виклик батьківського конструктора.
Синтаксис виклику конструктора батьківського класу:

```
super().__init__(self, <параметри>).
```

`super()` - це посилання на батьківський клас.
Альтернатива:

```
<ім'я базового класу>.__init__(self, <параметри>)
```

Приклад конструктора в класі `Seller` для додавання поля `cash`:

```
def __init__(self, name1='', age1=0, cash1=0 ):
        super().__init__(name1, age1)
        self.cash=cash1
```

Приклад 2. Успадкування конструктора

shopInherit1.py

PYTHON_BASICS > shopInherit1.py > Seller > __init__

```python
class ShopWorker:
    _count_workers=0
    def __init__(self, name1='', age1=0):
        self.name = name1
        self.__age = age1
        self.setting_id()
    def setting_id(self):
        ShopWorker._count_workers +=1
        self.id = ShopWorker._count_workers
    def add_year(self):
        self.__age+=1
    def __str__(self):
        str_out="Працівник "+str(self.id)+": "+self.name+" "+str(self.__age)
        str_out += " всіх працівників " + str(ShopWorker._count_workers)
        return str_out
    @staticmethod
    def info():
        print('В магазині працює: ', ShopWorker._count_workers, ' працівників')
    @classmethod
    def naming_shop(cls, name):
        cls.name_shop= name
        return cls.name_shop
    def get_age(self):
        return self.__age
    def set_age(self, new_age):
        if (new_age < 0):
            new_age= - new_age
        self.__age=new_age
```

Опис класу **ShopWorker**
Створення атрибуту класу **count_workers**
Конструктор класу

Виклик методу **setting_id** з конструктора
Опис методу **setting_id**
Ініціалізація полів **count_workers** та **id**

Опис методу **add_year** для збільшення віку об'єкта

Опис методу **__str__**

Статичний метод info()

Метод класу naming_shop(cls, name):

Гетер для атрибуту **__age**

Сетер для атрибуту **__age**
Перевірка параметра **new_age** на допустиме значення, та його коригування

Приклад 2. Успадкування конструктора

shopInherit1.py

PYTHON_BASICS > shopInherit1.py > ...

```
30  class Seller (ShopWorker):
31      def __init__(self, name1='', age1=0, cash1=0 ):
32          super().__init__(name1, age1)
33          self.cash=cash1
34
35  ShopWorker.info()
36
37  worker_one = ShopWorker('Іван', 25)
38  print (worker_one)
39  worker_one.add_year()
40  print (worker_one)
41  ShopWorker.naming_shop('Fara')
42  print ('Назва магазину: ',ShopWorker.name_shop)
43
44  seller1= Seller('Оксана',28, 3456.50)
45  print(seller1)
46  print(seller1.name)
47  print(seller1.cash)
48  seller2= Seller('Марія',40)
49  print(seller2)
50  print(seller2.name)
51  print(seller2.cash)
52  print(Seller.name_shop)
53  print(Seller._count_workers)
```

TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE JUPYTER Python

```
В магазині працює:  0  працівників
Працівник 1: Іван 25 всіх працівників 1
Працівник 1: Іван 26 всіх працівників 1
Назва магазину:  Fara
Працівник 2: Оксана 28 всіх працівників 2 працює продавцем з готівкою 3456.5
Оксана
3456.5
Працівник 3: Марія 40 всіх працівників 3 працює продавцем з готівкою 0
Марія
0
Fara
3
```

Опис класу **`Seller`**, що наслідує **`ShopWorker`**
Перевизначення конструктора батьківського класу

Вивід атрибуту класу **`ShopWorker`**

Створення об'єкту **`worker_one`**
Вивід даних про працівника **`worker_one`**
Виклик методу для збільшення віку **`add_year()`**
Повторний вивід даних про працівника **`worker_one`**
Виклик методу класу **`naming_shop`** через назву класу
Вивід атрибута класу **`name_shop`**

Створення об'єкту **`seller1`**
Вивід даних про працівника **`seller1`**
Вивід поля **`name`** для працівника **`seller1`**
Вивід поля **`cash`** для працівника **`seller1`**
Створення об'єкту **`seller2`**
Вивід даних про працівника **`seller2`**
Вивід поля **`name`** для працівника **`seller2`**
Вивід поля **`cash`** для працівника **`seller2`**
Вивід атрибута класу **`name_shop`** через назву класу **`Seller`**
Вивід атрибута класу **`_count_workers`** через назву класу **`Seller`**

Результат роботи коду

Успадкування методів

Дочірний клас **успадковує відкриті** методи батьківського класу.

В дочірньому класі можна додавати **власні методи** для збільшення функціональності класу.

Приватні методи базового класу **не успадковуються**.

В дочірньому класі можна **перевизначати** методи прописані в батьківському класі.

`super()` - посиланням на батьківський клас.

Успадкування методів

“Магічні” методи наслідуються класами від класу **`object`** та можуть бути перевизначені.

Метод **__str__** перевизначений в класі класі **ShopWorker**:

```
def __str__(self):
        str_out="Працівник "+str(self.id)+": "+self.name+" "+str(self.__age)
        str_out += " всіх працівників " + str(ShopWorker._count_workers)
        return str_out
```

Перевизначення методу **__str__** в класі **Seller**:

```
def __str__(self):
        return super().__str__()+ " працює продавцем з готівкою "+ str(self.cash)
```

Метод **`working`** в класі **ShopWorker**:

```
def working(self):
        print ('Виконую роботу')
```

Перевизначення методу **`working`** класі **Seller**:

```
def working(self):
        print ('Обслуговую покупців')
```

Приклад 3. Успадкування методів

shopInherit2.py

PYTHON_BASICS > shopInherit2.py > ...

```
1   class ShopWorker:
2       _count_workers=0
3       def __init__(self, name1='', age1=0):
4           self.name = name1
5           self.__age = age1
6           self.setting_id()
7       def setting_id(self):
8           ShopWorker._count_workers +=1
9           self.id = ShopWorker._count_workers
10      def working(self):
11          print('Виконую роботу')
12      def __str__(self):
13          str_out="Працівник "+str(self.id)+": "+self.name+" "+str(self.__age)
14          str_out += " всіх працівників " + str(ShopWorker._count_workers)
15          return str_out
16      @staticmethod
17      def info():
18          print('В магазині працює: ', ShopWorker._count_workers, ' працівників')
19      @classmethod
20      def naming_shop(cls, name):
21          cls.name_shop= name
22          return cls.name_shop
23      def get_age(self):
24          return self.__age
25      def set_age(self, new_age):
26          if (new_age < 0):
27              new_age= - new_age
28          self.__age=new_age
```

Опис класу **ShopWorker**
Створення атрибуту класу **count_workers**
Конструктор класу

Виклик методу **setting_id** з конструктора
Опис методу **setting_id**
Ініціалізація полів **count_workers** та **id**

Опис методу **working**

Опис методу **__str__**

Статичний метод info()

Метод класу naming_shop(cls, name):

Гетер для атрибуту **__age**

Сетер для атрибуту **__age**
Перевірка параметра **new_age** на допустиме значення, та його коригування

Приклад 3. Успадкування методів

shopInherit2.py

PYTHON_BASICS > shopInherit2.py > ...

```
30  class Seller (ShopWorker):
31      def __init__(self, name1='', age1=0, cash1=0 ):
32          super().__init__(name1, age1)
33          self.cash=cash1
34      def __str__(self):
35          return super().__str__()+ " працює продавцем з готівкою "+ str(self.cash)
36      def working(self):
37          print ('Обслуговую покупців')
38
39  ShopWorker.info()
40
41  worker_one = ShopWorker('Іван', 25)
42  print (worker_one)
43  worker_one.working()
44
45  print ('Назва магазину: ',ShopWorker.naming_shop('Fara'))
46
47  seller1= Seller('Оксана',28, 3456.50)
48  print(seller1)
49  seller1.working()
50  seller2= Seller('Марія',40)
51  print(seller2)
52  seller2.working()
53  print(Seller.name_shop)
54  print(Seller._count_workers)
```

TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE JUPYTER Python

```
В магазині працює:  0  працівників
Працівник 1: Іван 25 всіх працівників 1
Виконую роботу
Назва магазину:  Fara
Працівник 2: Оксана 28 всіх працівників 2 працює продавцем з готівкою 3456.5
Обслуговую покупців
Працівник 3: Марія 40 всіх працівників 3 працює продавцем з готівкою 0
Обслуговую покупців
Fara
3
```

Опис класу **Seller**, що наслідує **ShopWorker**
Перевизначення конструктора батьківського класу

Перевизначення методу **__str__** в класі **Seller.**

Перевизначення методу **working** в класі **Seller.**

Вивід атрибуту класу **ShopWorker**

Створення об'єкту **worker_one**
Вивід даних про працівника **worker_one**
Виклик методу **working() визначено в класі ShopWorker.**

Вивід атрибута класу **name_shop,** через виклик методу класу **naming_shop**

Створення об'єкту **seller1**
Вивід даних про працівника **seller1**
Виклик методу **working() визначено в класі Seller**
Створення об'єкту **seller2**
Вивід даних про працівника **seller2**
Виклик методу **working() визначено в класі Seller**
Вивід атрибута класу **name_shop** через назву класу **Seller**
Вивід атрибута класу **_count_workers** через назву класу **Seller**

Результат роботи коду

Лінеаризація

Лінеаризація — це черговість, при якій проводиться пошук зазначеного атрибута чи методу в ієрархії класів.

Лінеаризацію для певного класу можна отримати через конструкції:

- `<ім'я класу>.__mro__`
- `<ім'я класу>.mro()`

Лінеаризація має назву **MRO — "Method Resolution Order"** (порядок вирішення методів) - вона застосовується як для методів так і атрибутів.

В Python є вбудована функція: `isinstance(obj, cls)`, яка повертає `True` якщо об'єкт `obj` є екземпляром класу `cls` або його суперкласів.

Множинне наслідування

Множинне наслідування— властивість деяких мов програмування, в яких класи можуть успадкувати поведінку і властивості більш ніж від одного суперкласу безпосередньо.

Найпростіший приклад:

```
class Horse:
    def say_hello(self):
       print("Я - кінь!")
class Eagle:
    def say_hello(self):
        print("Я - орел!")
class Pegasus(Horse, Eagle):
         pass

p = Pegasus()
p.say_hello()
```

Перевага множинного наслідування - дозволяє проектувати доволі складні і гнучкі ієрархії класів.

Недолік множинного наслідування - використання цього механізму може призвести до появи у коді доволі серйозних помилок.

Множинне наслідування на практиці використовується вкрай рідко.

Підсумки

- **Наслідування** - це властивість, що дозволяє описати новий клас на основі вже наявного з частковим чи повним збереженням функціональності.
- **Базовий, суперклас або батьківський** - це клас, від якого наслідується новий.
- Новий клас, що наслідує батьківський навають - **нащадком, підкласом** або **дочірнім класом.**
- Дочірній клас **успадковує відкриті** (публічні) атрибути та методи базового класу.
- У дочірньому класі **приватні** атрибути та методи базового класу **не успадковуються.**
- **object** - це базовий клас для всіх класів, бо від цього класу явно або неявно успадковуються всі інші класи