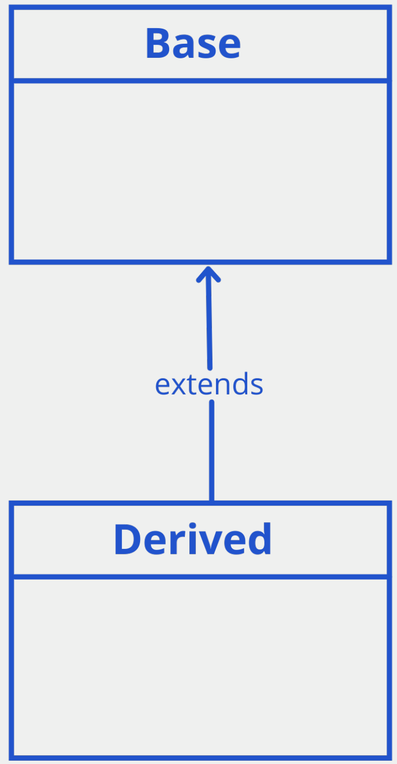
успадкування: визначення, терміни

Успадкування - одне з ключових понять ООП і є потужним інструментом при написанні коду. Класи які ви створюєте (похідні), можуть успадковуватись від інших класів (батьківських), **успадковуючи властивості батьківських класів** і **додавати свої властивості і поведінку**.



Успадкування моделює **зв'язок**. Це означає що **Derived** клас, **який успадковує Base** клас, зв'язаний з Base наслідуванням і **є спеціалізованою версією класу Base.**

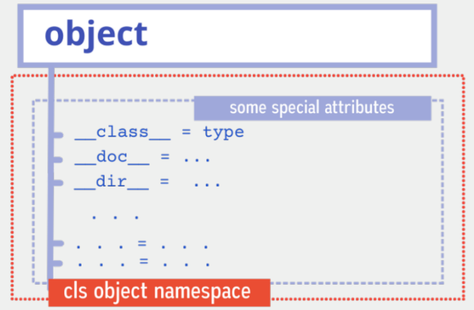
Для зображення відношень між класами часто використовують **UML** (**Unified Modeling Language**) - яка використовує графічні позначення для створення моделей систем (не лише класів, вона використовується значно ширше).

Термінологія:

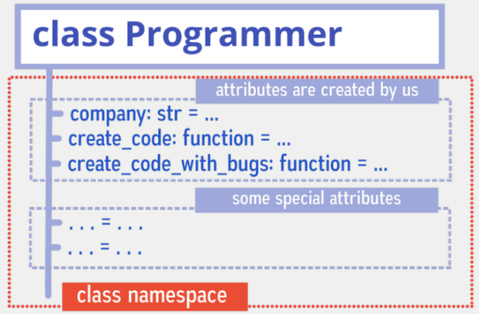
- Класи, які успадковуються від інших, називаються **похідними класами**, **дочірніми класами**, **підкласами** або **підтипами**.
- Класи, з яких походять інші класи, називаються **базовими класами**, **батьківськими** або **суперкласами**.
- Кажуть, що похідний клас успадковує або розширює базовий клас.

успадкування: перші кроки

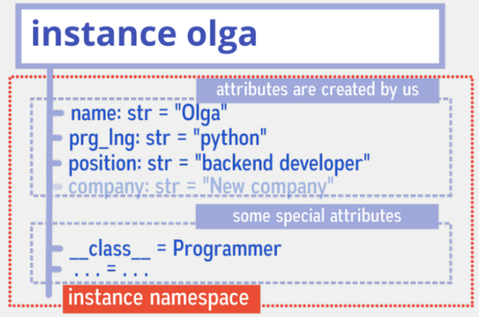
Ми з Вами вже використовували наслідування, але просто про це не знали). Згадайте малюнок, на основі якого ми розбирали шлях пошуку атрибутів екземпляру - і зрозуміли що всі загальні для всіх екземплярів атрибути зберігаються не в екземплярах, а в класі. Щоб розібрати цю ситуацію глибше, згадаємо функцію dir(obj), яка повертає список всіх атрибутів об’єкта.




```
>>> class MyClass:
>>>     pass
...
>>> dir(MyClass)
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__',
'__getattribute__', '__getstate__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__le__',
'__lt__', '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__',
'__sizeof__', '__str__', '__subclasshook__', '__weakref__']
```



зверніть увагу на кількість “спеціальних” атрибутів в пустому класі. Звідки вони? Справа в тому, що у python існує спеціальний клас, від якого наслідуються всі інші класи. Цей клас називається **object** (саме так, з маленької букви щоб підкреслити його особливість) і саме він надає класам майже всі ці “спеціальні” атрибути (трохи інакше лише для виключень, про це пізніше):




```
>>> dir(object)
['__class__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__',
'__getstate__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '__ne__', '__new__',
'__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__']
```

Тобто, шлях пошуку атрибуту: в екземплярі, якщо немає - в класі, немає - у батьківському класі.

успадкування: синтаксис

Синтаксис, який ми можемо використовувати для того щоб явно вказати що ми наслідуємо якийсь клас, для попереднього випадку виглядає так:

```
>>> class
MyClass(object):
    pass
...
...
>>>
```

тобто, при визначенні класу **MyClass** ми в дужках вказуємо ім'я класу від якого ми наслідуємо цей клас.
Звертаю увагу - вказувати клас object - не треба, це і так буде зроблено, якщо нічого не буде вказано іншого.

Згадаємо наш клас **Programer**, який ми вже використовували раніше:

```
>>> class Programmer:
        company = "Web factory"
        def __init__(self, name,
prg_lng):
            self.name = name
            self.prg_lng = prg_lng
        def create_code(self):
            return "awesome code"
        def create_bug(self):
            return "code with bugs"
...
...
>>>
```

Тепер давайте уявимо що нам треба описати клас **TecnLead** - який окрім того що є програмістом - з усіма його атрибутами, має ще додаткову поведінку - вміє організувати роботу команди.

```
>>> class TechLead(Programmer):
    def manage_team(self):
        return f"{self.name} actively manages the development team of
{self.company}"

...
>>>
```

Щоб не повторювати все те, що вже є у класі **Programer**, ми можемо наслідувати клас **TechLead** від нього, визначивши лише притаманні лише цьому класу атрибути - тобто створити спеціалізовану версію класу **Programer** (тому що **TechLead** - залишається і програмістом також).