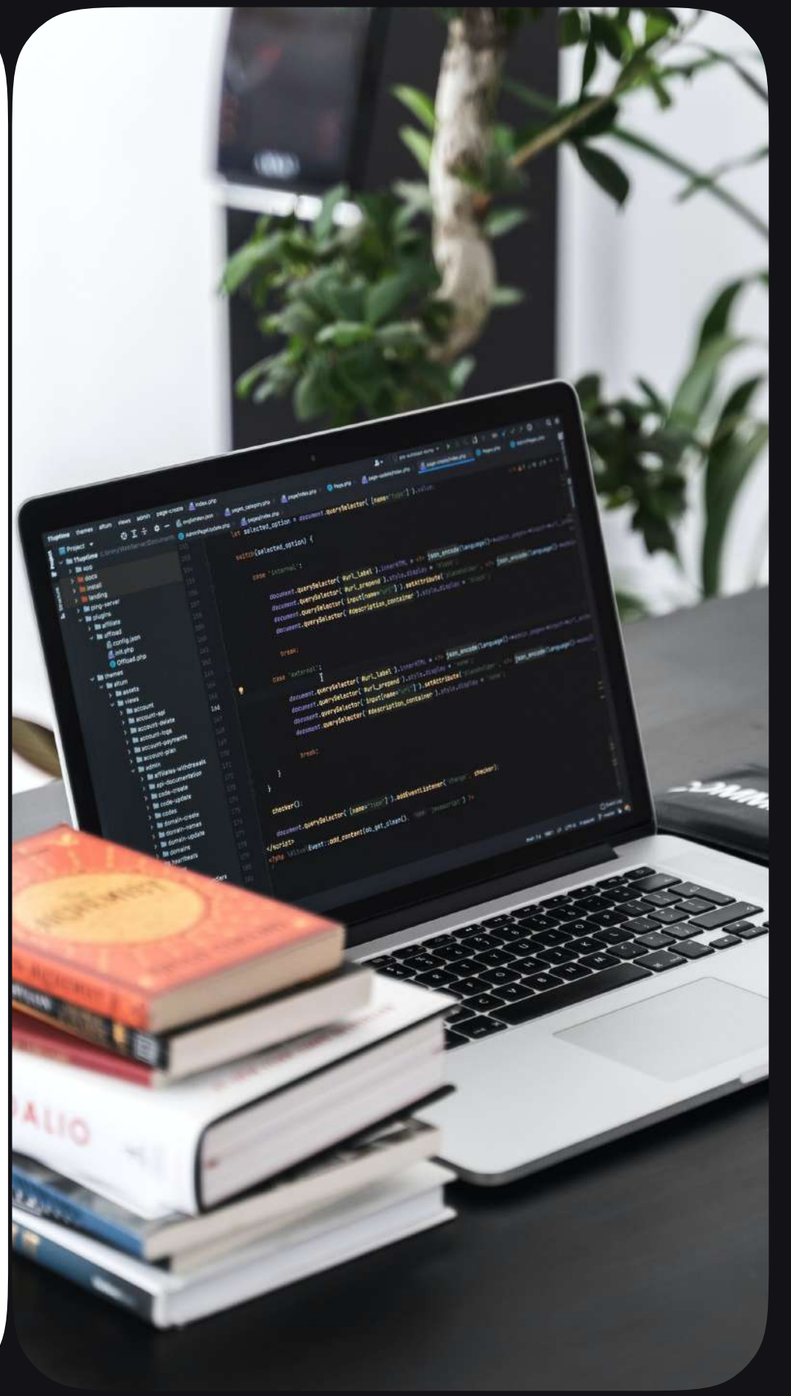
Модуль 2 Занятие 6

Наследование



Наследование

Базовый
и производный класс

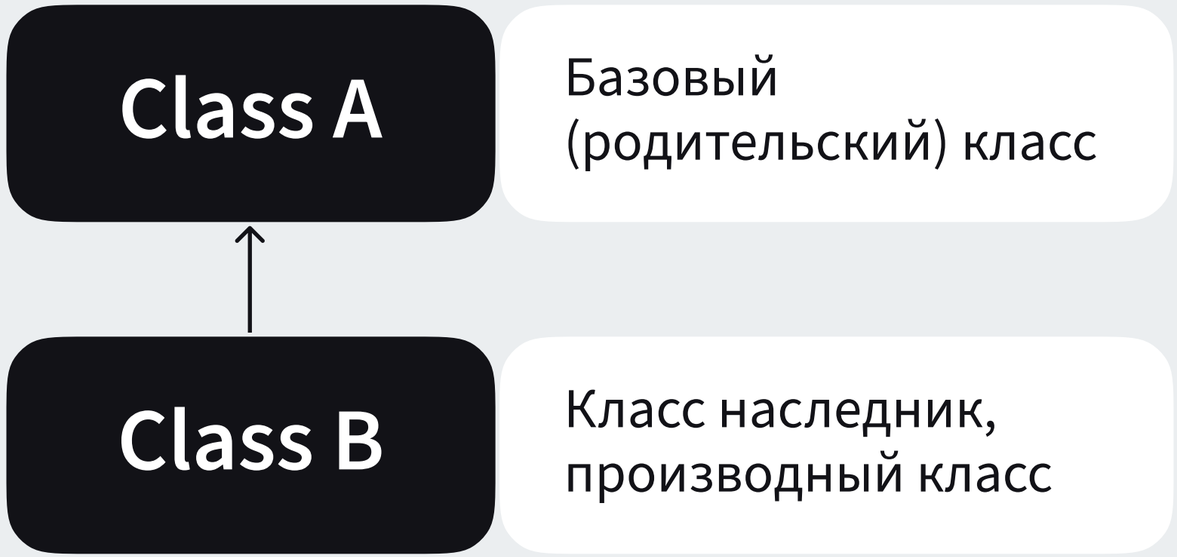
Наследование
от object

super()

Множественное
наследование

Наследование

Наследование — это принцип ООП, согласно которому возможно использование **базового класса**, описывающего общие свойства и методы для других классов. Такие классы называют классы наследники или производные классы.



Пример

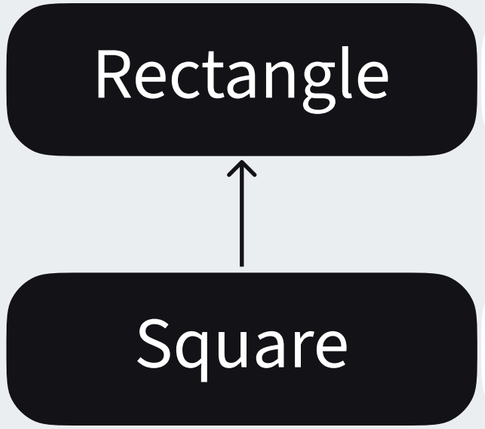
```python
class Square:
    def __init__(self, a):
        self.a = a

    def __str__(self):
        return f'Квадрат со стороной {self.a}'

    def get_area(self):
        return self.a * self.a

class Rectangle:
    def __init__(self, a, b):
        self.a = a
        self.b = b

    def __str__(self):
        return f'Прямоугольник со сторонами {self.a} {self.b}'

    def get_area(self):
        return self.a * self.b
```

Square	Rectangle

def _init_(...):
...
def _str_(...):
...
def get_area(...):
...
	def _init_(...):
	...
	def _str_(...):
	...
	def get_area(...):
	...

Наследование в Python

```python
class Rectangle:
    def __init__(self, a, b):
        self.a = a
        self.b = b

    def __str__(self):
        return f'Прямоугольник со сторонами {self.a} {self.b}'

    def get_area(self):
        return self.a * self.b

class Square(Rectangle):
    def __str__(self):
        return f'Квадрат со стороной {self.a}'

square = Square(5, 5)
rec = Rectangle(5, 6)
print(square, square.get_area())
print(rec, rec.get_area())
```



Базовый класс

Класс наследник

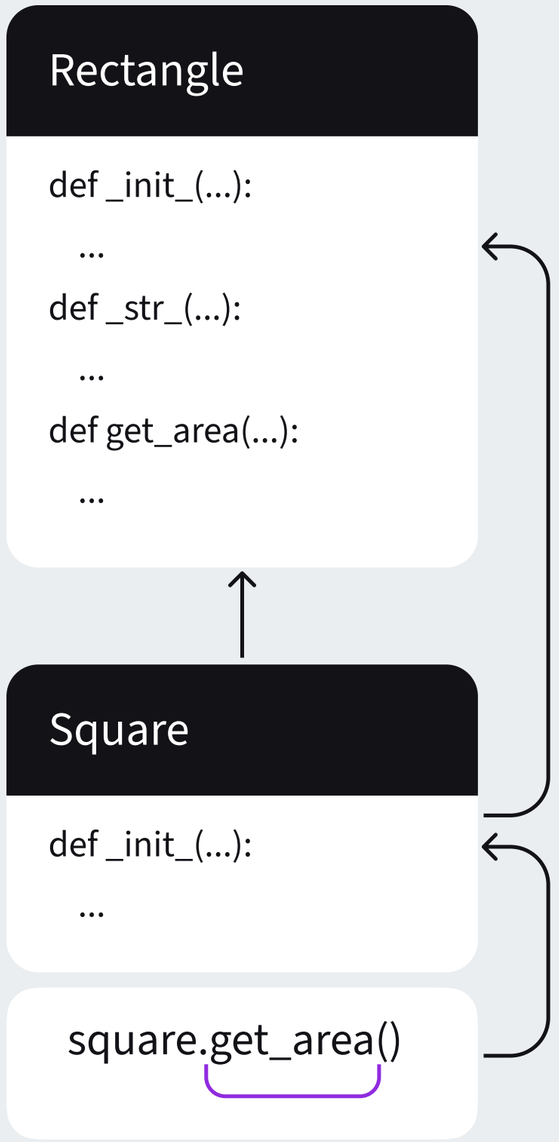
Вывод:

```
25
30
```

Как происходит наследование

```
class Rectangle:
    def __init__(self, a, b):
        self.a = a
        self.b = b

    def __str__(self):
        return f'Прямоугольник со сторонами {self.a} {self.b}'

    def get_area(self):
        return self.a * self.b

class Square(Rectangle):
    def __str__(self):
        return f'Квадрат со стороной {self.a}'

print(Square.__dict__)
print(Rectangle.__dict__)
```

Посмотрите, что будет выведено на экран?



Переопределение и расширение

Базовый класс для класса Square

Переопределение метода object, а также метод переопределен в классе Square

Переопределение метода класса Rectangle

Расширение базового класса

```python
class Rectangle:
    def __init__(self, a, b):
        self.a = a
        self.b = b

    def __str__(self):
        return f'Прямоугольник со сторонами {self.a} {self.b}'

    def get_area(self):
        return self.a * self.b


class Square(Rectangle):
    def __str__(self):
        return f'Квадрат со стороной {self.a}'

    def get_perimeter(self):
        return self.a * 4
```

Вызов метода базового класса

```
class Rectangle:
    def __init__(self, a, b):
        self.a = a
        self.b = b

    def __str__(self):
        return f'Прямоугольник со сторонами {self.a} {self.b}'

    def get_area(self):
        return self.a * self.b

class Square(Rectangle):
    def __init__(self, a):
        self.a = a

    def __str__(self):
        return f'Квадрат со стороной {self.a}'

square = Square(5)
print(square, square.get_area())
```

Ошибка:

```
AttributeError: 'Square' object has no attribute 'b'
```

У экземпляра класса Square только один атрибут — **a**. Можно выйти из данной ситуации, если каким-то образом вызвать инициализатор родительского класса и передать туда два равных параметра.

Вызвать метод базового класса можно с помощью функции **super()**.

Функция super()

super() — возвращает ссылку на объект посредник, который позволяет получить доступ к методам базового класса.

Основное применение функции **super()** — получение доступа из класса наследника к методам родительского класса, в случае если эти методы были переопределены в классе-наследнике.

Пример

```
class Rectangle:
    def __init__(self, a, b):
        self.a = a
        self.b = b

    def __str__(self):
        return f'Прямоугольник со сторонами {self.a} {self.b}'

    def get_area(self):
        return self.a * self.b

class Square(Rectangle):
    def __init__(self, a):
        super().__init__(a, a)

    def __str__(self):
        return f'Квадрат со стороной {self.a}'

square = Square(5)
rec = Rectangle(5, 6)
print(square, square.get_area())
print(rec, rec.get_area())
```

Вызов метода базового класса

super().__init__(a, a)

можно записать также как:

Rectangle.__init__(self, a, a)

В этом случае мы явно указываем класс, метод и первым параметром передаем ссылку на экземпляр класса.

Вывод:

```
25
30
```

Пример

```
class Point2D:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __str__(self):
        return f'Точка ({self.x}, {self.y})'

class Point3D(Point2D):
    def __init__(self, x, y, z):
        super().__init__(x, y)
        self.z = z

    def __str__(self):
        return f'Точка ({self.x}, {self.y}, {self.z})'

point1 = Point2D(1, 2)
point2 = Point3D(1, 2, 3)
print(point1)
print(point2)
```

В этом примере мы добавим атрибуты **x** и y с помощью инициализатора родительского класса, а затем добавим еще один атрибут **z**.

Вывод:

```
Точка (1, 2)
Точка (1, 2, 3)
```

Наследование от object

```
class Test:
    pass


test1 = Test()
test2 = Test()
print(test1)
print(test1 == test2)
```

```
class Test(object):
    pass
```

Вывод:

```
<__main__.Test object at 0x000002DB9AB65450>
False
```

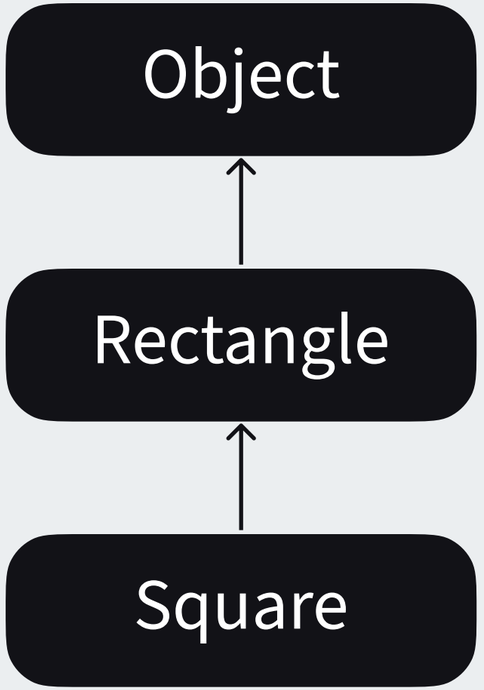
Object

class,
delattr,
dir,
doc,
_eq_,
format,
_ge_,
getattribute,
_gt_,
hash,
init,
_init_subclass_,
и так далее…

Test

Наследование от object

```
class Rectangle:
    def __init__(self, a, b):
        self.a = a
        self.b = b

    def __str__(self):
        return f'Прямоугольник со сторонами {self.a} {self.b}'

    def get_area(self):
        return self.a * self.b

class Square(Rectangle):
    def __init__(self, a):
        super().__init__(a, a)

    def __str__(self):
        return f'Квадрат со стороной {self.a}'
```



issubclass()

```
class Rectangle:
    pass

class Square(Rectangle):
    pass

print(issubclass(Rectangle, object))
print(issubclass(Square, object))
print(issubclass(Square, Rectangle))
print(issubclass(Rectangle, Square))
```

Вывод:

```
True
True
True
False
```

Функция issubclass позволяет проверить, является ли класс наследником другого класса.

Функция работает только с классами и не работает с объектами.

isinstance()

```
class Rectangle:
    pass

class Square(Rectangle):
    pass

square = Square(5)
rectangle = Rectangle(5, 6)
print(isinstance(rectangle, object))
print(isinstance(square, object))
print(isinstance(square, Rectangle))
print(isinstance(rectangle, Square))
```

Функция isinstance позволяет проверить, является ли объект экземпляром класса, либо экземпляром наследующегося от него класса.

Вывод:

```
True
True
True
False
```

Стандартные типы

Стандартные типы в Python — тоже классы

```
print(issubclass(int, object))
print(issubclass(list, object))
```

Вывод:

```
True
True
```

Так как стандартные типы — классы,
то можно выполнять наследование
от стандартных типов и расширить их.

Пример

```
class MyList(list):
    def get_avg(self):
        return sum(self) / len(self)


lst = MyList()
for i in range(1, 11):
    lst.append(i)
print(lst.get_avg())
```

Вывод:

```
5.5
```

Расширяем базовый класс методом get_avg(), который будет возвращать среднее арифметическое элементов списка.

Пример

```
class MyList(list):
    def __init__(self, name):
        super().__init__(self)
        self.name = name

    def __str__(self):
        result = super().__str__()
        return f'{self.name}: {result}'

lst = MyList('Список нечетных чисел')
for i in range(1, 11, 2):
    lst.append(i)
print(lst)
```

Вывод:

```
Список нечетных чисел: [1, 3, 5, 7, 9]
```

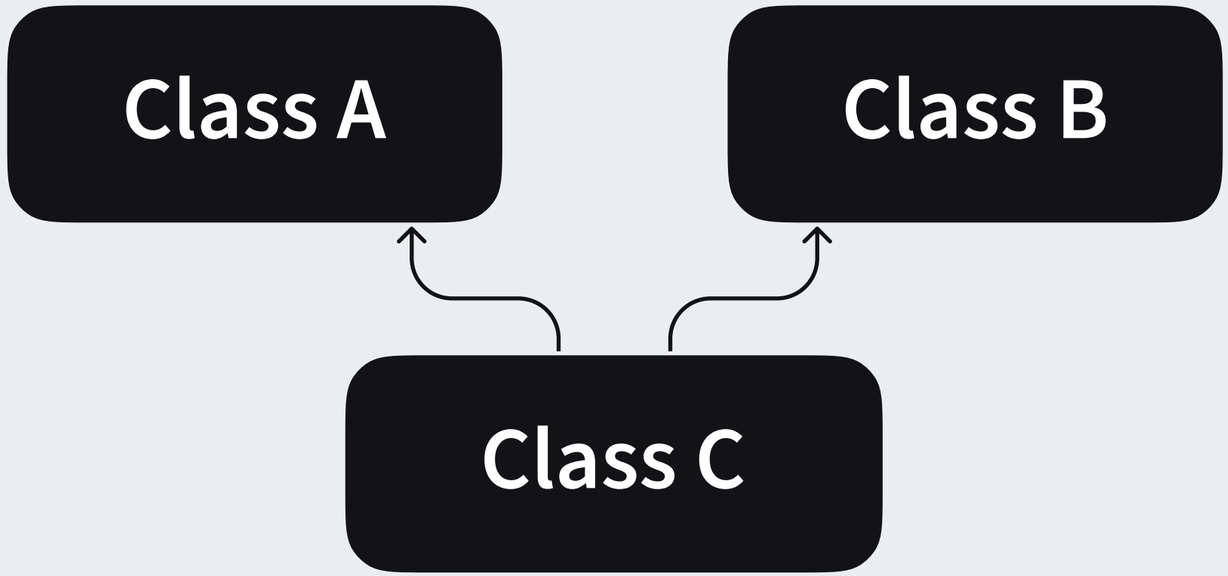
Переопределяем методы __init__ и __str__, при этом используем методы родительского класса с помощью функции super().

Множественное наследование

Множественное наследование — это возможность класса иметь несколько родительских классов.

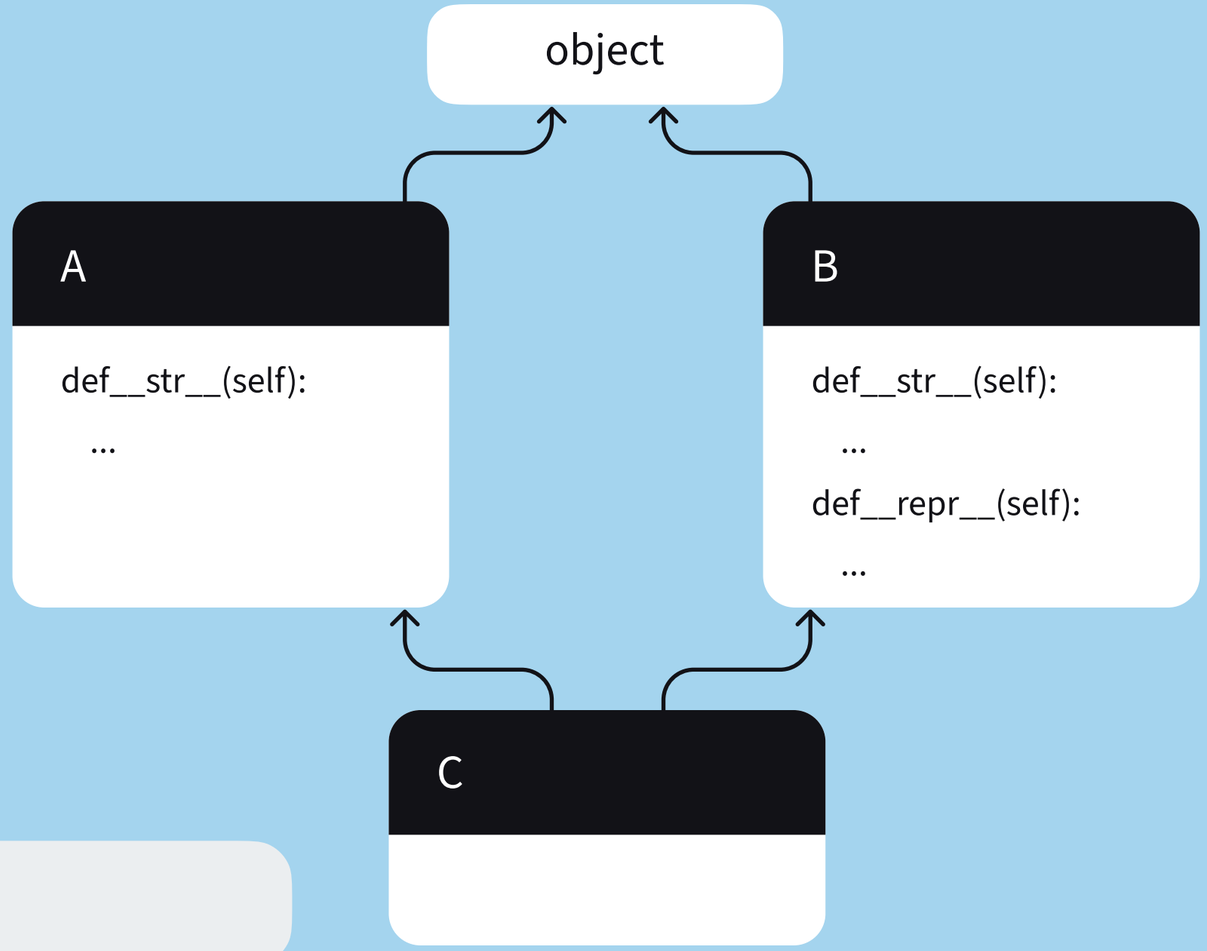
При множественном наследовании класс наследник наследует все атрибуты родительских классов.

Базовые (родительские) классы



Пример

```
class A:
    def __str__(self):
        return 'Экземпляр класса A'

class B:
    def __str__(self):
        return 'Экземпляр класса B'

    def __repr__(self):
        return 'B()'

class C(A, B):
    pass

c = C()
lst = [c]
print(lst[0])
print(lst)
```



Вывод:

```
Экземпляр класса A
[B()]
```

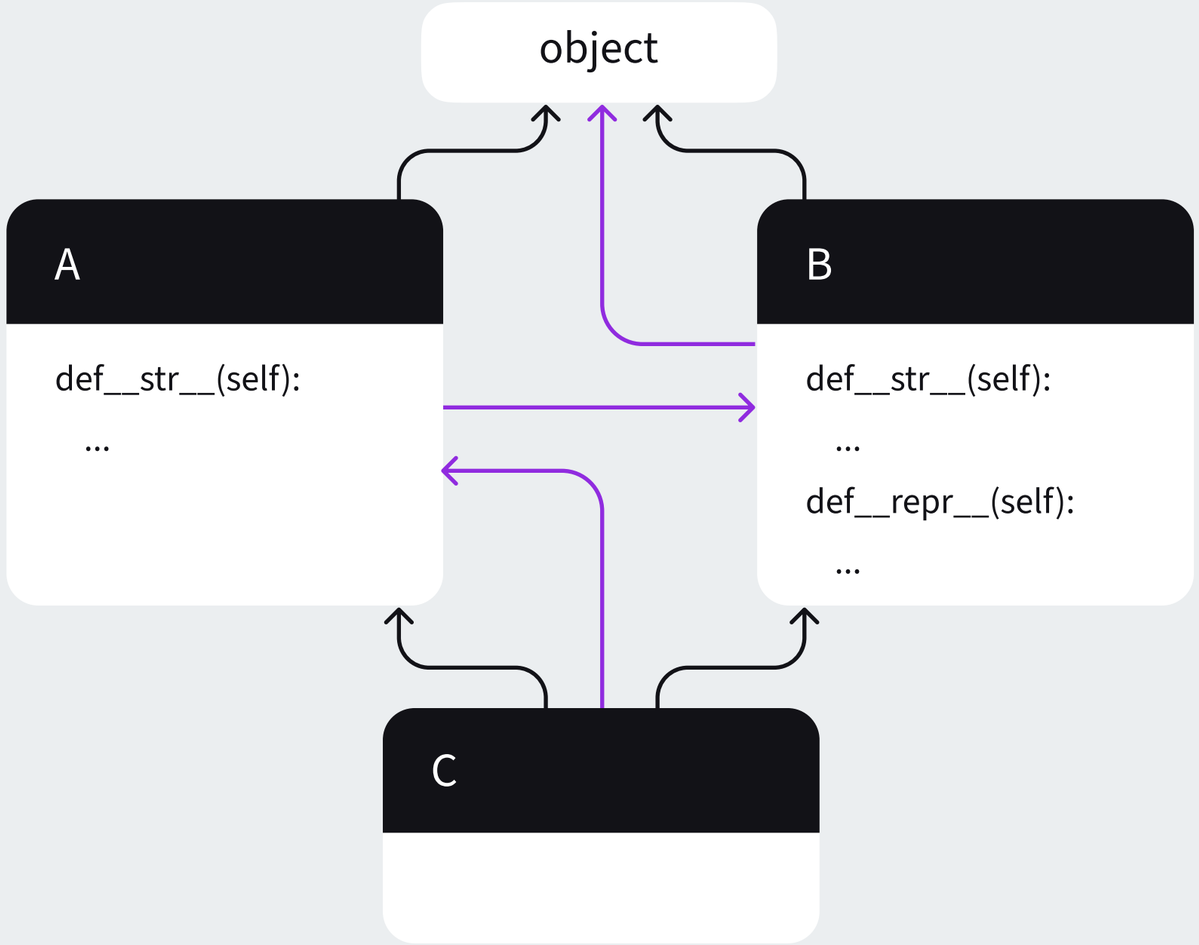
MRO

MRO — порядок разрешения методов.
Порядок разрешения методов позволяет выяснить, из какого родительского класса нужно использовать метод, если он не найден в классе наследнике.

Вывести порядок разрешения методов для класса можно с помощью специальной переменной **__mro__**.

MRO

```
class A:
    pass

class B:
    pass

class C(A, B):
    pass

print(C.__mro__)
```



Вывод:

```
(<class '__main__.C'>, <class '__main__.A'>, <class '__main__.B'>, <class 'object'>)
```

Итоги

- Наследование — это использование базового (родительского) класса для других классов — классов наследников
- class B(A): … — создание класса B наследуемого от класса A
- Функция super() позволяет получить доступ из класса наследника к методам родительского класса
- Все пользовательские классы наследуются от object
- Функция issubclass() позволяет проверить, является ли класс наследником другого класса
- Функция isinstance() позволяет проверить, является ли объект экземпляром класса, либо экземпляром наследующегося от него класса
- Стандартные типы в Python — классы и от них можно наследоваться
- MRO — порядок разрешения методов при множественном наследовании. Можно получить, используя специальную переменную __mro__