Классы Python

Суханов А.Я.

Классы

```
class Figure():
    def square():
        pass
    def perimeter():
        pass
class Rectangle(Figure):
    def __init__(self,left,top,width,height):
        self.top = top
        self.left = left
        self.width = width
        self.height = height
    def square(self):
        return self.width*self.height
    def perimeter(self):
        return (self.width+self.height)*2
```

```
class Rect(Rectangle):
    bottom = 0
    right = 0
    def __init__(self,left,top,right,bottom):
        #self.bottom = bottom
        #self.right = right
        super().__init__(left,top,right-left,bottom-top)
        pass
```

```
rect = Rectangle(0,0,10,10)
print(rect.square()) # 100

rect1 = Rect(10,10,20,20)
print(rect1.square()) # 100
rect2 = Rect(15,15,20,20)
print(rect2.square()) # 25

rect1.bottom = 1
rect1.right = 2

print(rect2.bottom) # 0
print(rect2.right) # 0
Rect.bottom = 3
Rect.right = 3

print(rect1.bottom)# 1
print(rect1.right) # 2

print(rect2.bottom) # 3
print(rect2.right) # 3
```

Вызов методов базового класса

- Надо вызвать метод базового класса из метода, который переопределен в производном классе.
- Из конструктора дочернего класса нужно явно вызывать конструктор родительского класса.
- Обращение к базовому классу происходит с помощью super()

Видно, что без явного вызова конструктора класса A не вызывается A.__init__ и не создается поле x класса A.

```
class A(object):
    def __init__(self, x=5):
        print('A.__init__')
        self.x = x

class B(A):
    def __init__(self, y=2):
        print('B.__init__')
        self.y = y

k = B(7)                    # B.__init__

print('k.y =', k.y)         # k.y = 7
#print('k.x =', k.x)        # AttributeError: 'B' object
has no attribute 'x'
```

Реализуем явный вызов

```
class A(object):
    def __init__(self, x=5):
        print('A.__init__')
        self.x = x

class B(A):
    def __init__(self, y=2):
        print('B.__init__')
        super().__init__(y/2)
        self.y = y

k = B(7)                        # B.__init__
                                # A.__init__
print('k.y =', k.y)             # k.y = 7
print('k.x =', k.x)             # k.x = 3.5
```

- Конструктор базового класса стоит вызывать раньше, чем инцилизировать поля класса-наследника, потому что поля наследника могут зависеть (быть сделаны из) полей экземпляра базового класса.

super() или прямое обращение к классу?

Метод класса можно вызвать, используя синтаксис вызова через имя класса:

```
class Base(object):
    def __init__(self):
        print('Base.__init__')

class A(Base):
    def __init__(self):
        Base.__init__(self)
        print('A.__init__')

k = A()                          #
Base.__init__
                                 # A.__init__
```

Происходит лишний вызов

```
class A(Base):
    def __init__(self):
        Base.__init__(self)
        print('A.__init__')

class B(Base):
    def __init__(self):
        Base.__init__(self)
        print('B.__init__')

class C(A, B):
    def __init__(self):
        A.__init__(self)
        B.__init__(self)
        print('C.__init__')

x = C()
```

Видно, что конструктор Base.__init__ вызывается дважды. Иногда это недопустимо (считаем количество созданных экземпляров класса, увеличивая в конструкторе счетчик на 1; выдаем очередное auto id какому-то нашему объекту, например, номер пропуска или паспорта или номер заказа).

```
# Base.__init__
# A.__init__
# Base.__init__ - второй вызов
# B.__init__
# C.__init__
```

```
class Base(object):
    def __init__(self):
        print('Base.__init__')
class A(Base):
    def __init__(self):
        super().__init__()
        print('A.__init__')
class B(Base):
    def __init__(self):
        super().__init__()
        print('B.__init__')
class C(A, B):
    def __init__(self):
        super().__init__()
        print('C.__init__')
x = C()
конструкторы обоих базовых классов
```

Для реализации наследования питон ищет вызванный атрибут начиная с первого класса до последнего. Этот список создается слиянием (merge sort) списков базовых классов:
дети проверяются раньше родителей.
если родителей несколько, то проверяем в том порядке, в котором они перечислены.
если подходят несколько классов, то выбираем первого родителя.

При вызове super() продолжается поиск, начиная со следующего имени в MRO. Пока каждый переопределенный метод вызывает super() и вызывает его только один раз, будет перебран весь список MRO и каждый метод будет вызван только один раз.

```
# Base.__init__
# B.  init__ - вызваны
# A.__init__ - порядок вызова
# C.__init__
```

Вызов метода дедушки

```
class A(object):
    def spam(self):
        print('A.spam')
class B(A):
    pass
class C(B):
    def spam(self):
        super().spam()
        print('C.spam')

y = C()
y.spam()
print(C.__mro__)
```

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

Декораторы методов.

```
>>> def method_friendly_decorator(method_to_decorate):
...     def wrapper(self, lie):
...         lie -= 3
...         return method_to_decorate(self, lie)
...     return wrapper
...
>>> class Lucy:
...     def __init__(self):
...         self.age = 32
...     @method_friendly_decorator
...     def sayYourAge(self, lie):
...         print(
"Мне {} лет, а ты бы сколько дал?".format(self.age + lie))
...
>>> l = Lucy()
>>> l.sayYourAge(-3)
Мне 26 лет, а ты бы сколько дал?
```