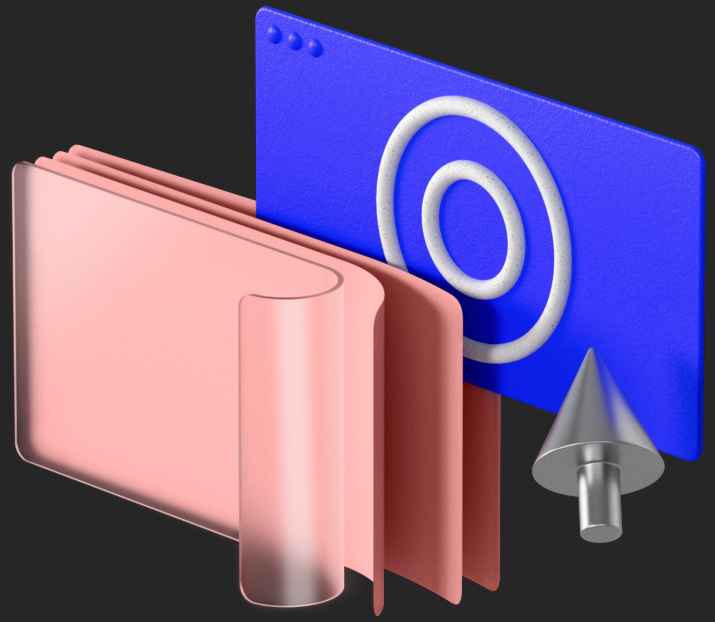
ООП



Андреев Роман

Парадигмы программирования

Императивный подход

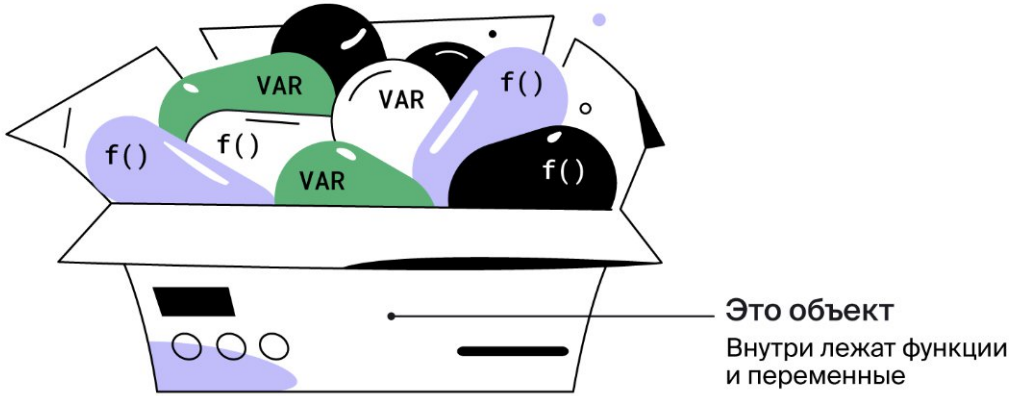
```
# Задаём список чисел.
numbers: list = [1, 2, 3, 4, 5]
# Резервируем ячейку памяти для результата.
result: int = 0
# Описываем, как нужно складывать элементы списка.
for i in numbers:
    result += i
print(result)
```

Декларативный подход, а именно — функциональный

```
# Описываем результат, который хотим получить.
def number_sum(data: list) -> int:
    result: int = 0
    for i in data:
        result += i
    return result

# Получаем результат.
print(number_sum([1, 2, 3, 4, 5]))
```

Объектно-ориентированное программирование (ООП)

Программисты договорились, что данные внутри объекта будут называть **свойствами** (или полями), а функции — **методами**.
Свойства представляют собой характеристики объекта, а методы — действия, которые умеет выполнять объект.



Создание классов в Python

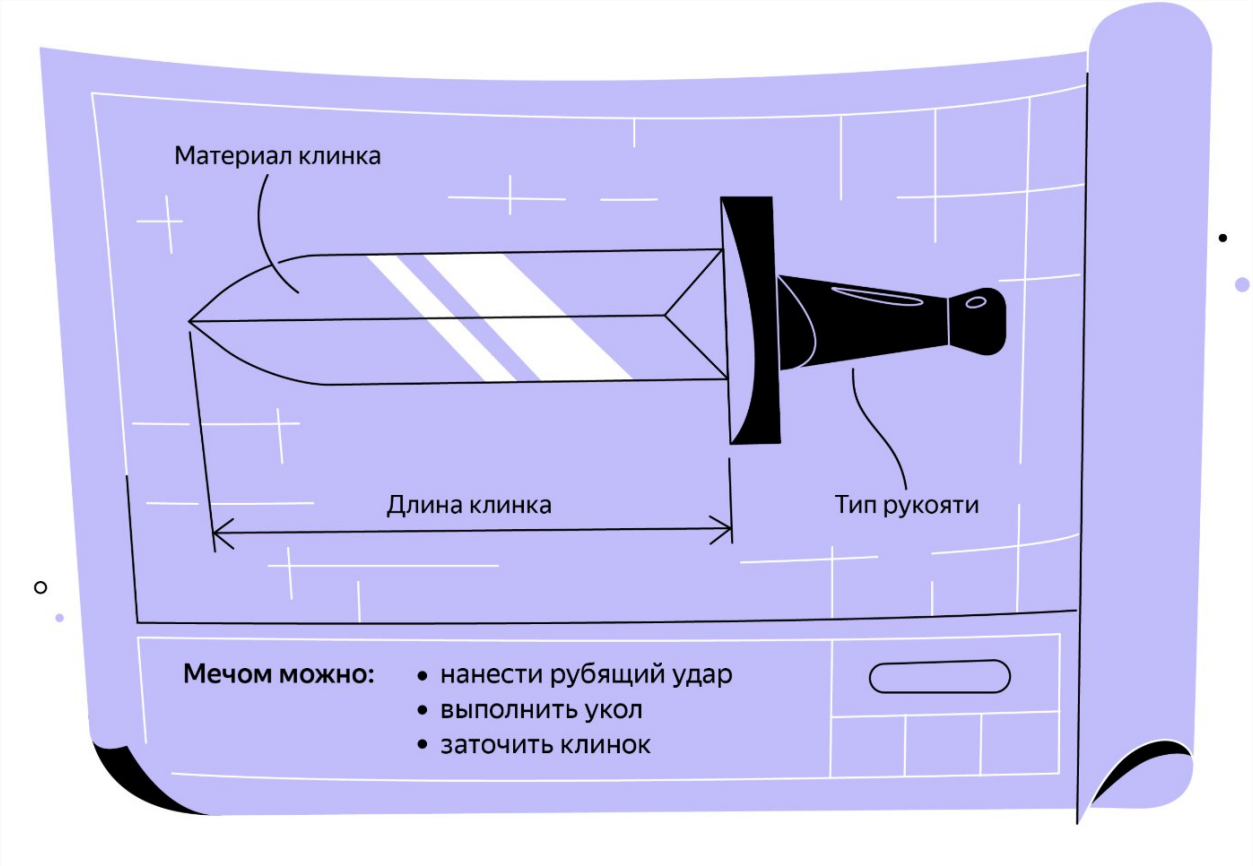
```
class <Имя класса>:
    # Конструктор класса – функция с именем __init__.
    def __init__(self, param1): # Параметры конструктора класса.
        # Объявление атрибутов.
        self.atr = param1
    # Объявление метода.
    def metod(self):
        # Код метода.
```

- имя класса должно начинаться с большой буквы;
- если имя класса состоит из нескольких слов, то используется *CamelCase*, то есть каждое слово пишется с большой буквы, между словами не используются пробелы, например TomReddl;

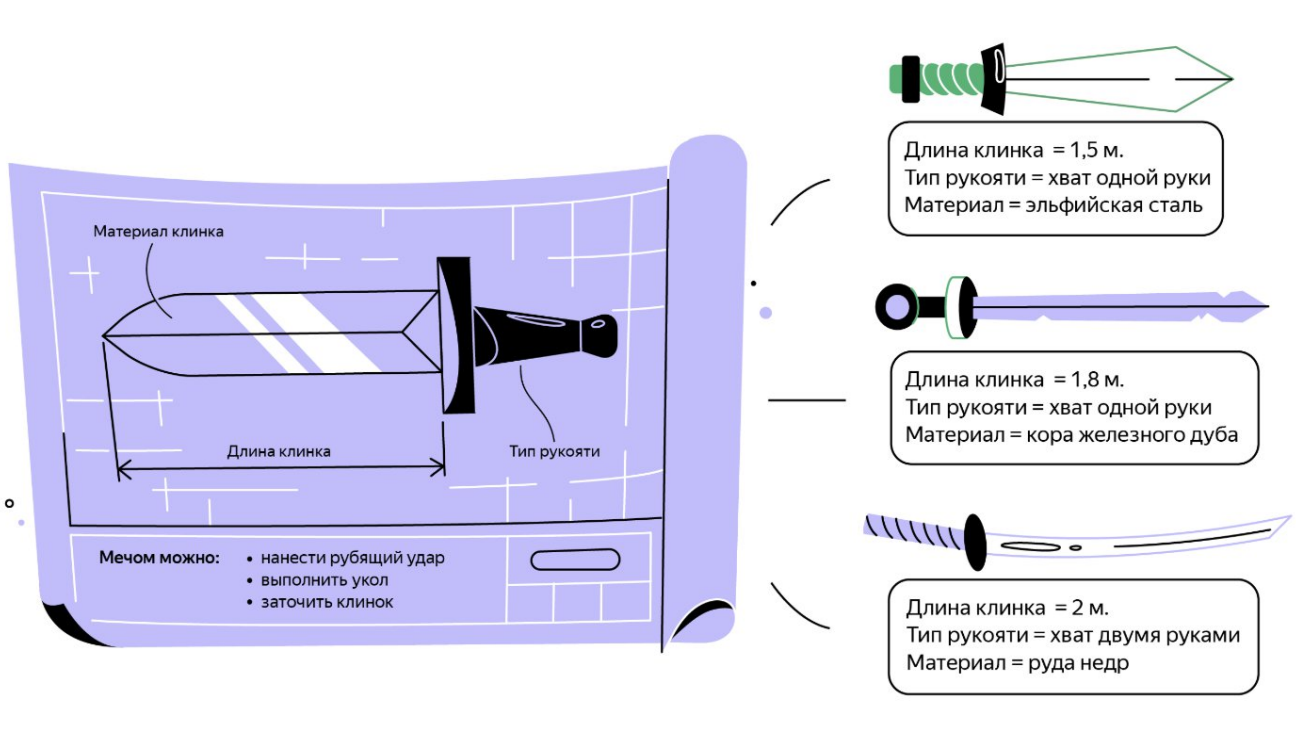
- имя класса должно быть в единственном числе, например Car, но не Cars.

Пишем первый класс

https://replit.com/join/yozuybdiuy-randreev



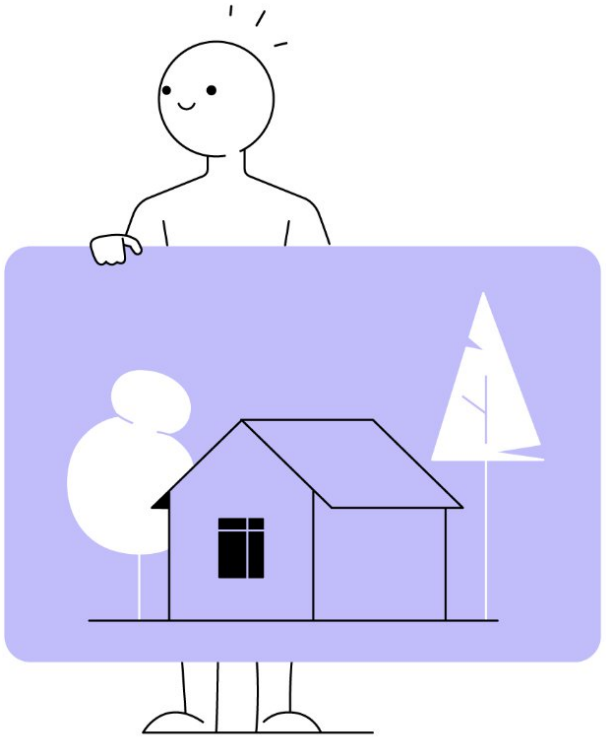
Объекты в Python



Класс (читай — эскиз) создан, теперь можно создать объекты.

Абстракция

Абстракция — это использование только значимых характеристик объекта и игнорирование всего остального.



Чем меньше характеристик, тем лучше абстракция, но ключевые характеристики убирать нельзя.

Интерфейс класса

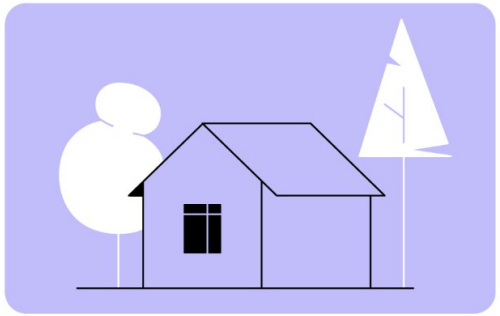
Интерфейс класса — это функциональная часть класса. В ООП интерфейсами называют свойства и методы класса, используя которые можно взаимодействовать с объектом этого класса из любого места в программе.

```
class Sword:
    #  Это конструктор, он вызывается при создании объекта.
    def __init__(self, name):
        self.name = name


    def show(self):
    # Это интерфейс класса, к нему можно обратиться из внешнего кода.
        print(f'Вы достали из ножен меч "{self.name}".')

# Создание объекта.
sparrow = Sword('Игла')
# Теперь можно воспользоваться его внешним интерфейсом: методом show().
sparrow.show()

# В терминал выведется:
# Вы достали из ножен меч Игла.
```

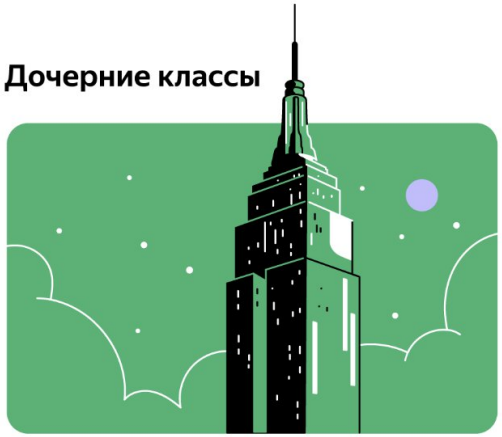
Наследование

Наследование — возможность описать новый класс на базе существующего.
При этом дочерние классы могут заимствовать свойства и методы родительского класса.

Родительский класс



Дочерние классы



Небоскрёб

Землянка хоббита

Еще пару терминов

Инкапсуляция — объединение и скрытие методов и свойств, и предоставление доступа к ним через простой внешний интерфейс.

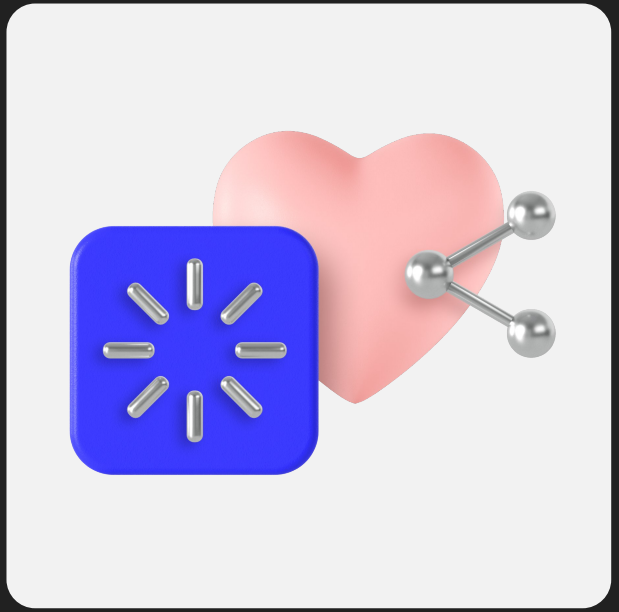
Даже не имея понятия, как работают встроенные методы lower, upper или split объекта типа str, вы из документации можете узнать, как с их помощью происходит управление объектами. Методы «инкапсулированы», разработчику предоставлен интерфейс для их вызова: string.upper().

Полиморфизм — возможность взаимодействовать с объектами разных классов через одинаковые интерфейсы, обращаться к свойствам и методам, общим для всех объектов.

*К какому наследнику класса MeleeWeapon вы ни обратились бы через свойство name или метод slashing_blow() — вы получите ответ (или, как минимум, не столкнётесь с ошибкой), потому что в Python реализован принцип **полиморфизма**: у всех наследников класса MeleeWeapon есть эти интерфейсы (свойство name и метод slashing_blow()), наследуемые или переопределённые.*

Практика

Перейдем к выполнению разбора практических задач



Вопросы и ответы