GeekBrains

Погружение в Python

Урок 10
ООП. Начало

Содержание урока

План курса

1. Основы Python
2. Простые типы данных
3. Коллекции
4. Функции
5. Итераторы и генераторы
6. Модули
7. Файлы и файловая система
8. Сериализация
9. Декораторы
10. ООП. Начало
11. ООП. Особенности Python
12. ООП. Финал
13. Исключения
14. Основы тестирования
15. Обзор стандартной библиотеки Python

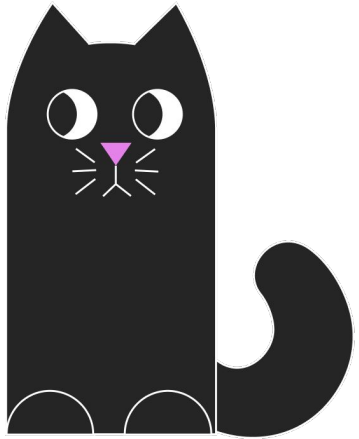
Что будет на уроке сегодня

- 📌 Разберёмся с объектно-ориентированным программированием в Python.
- 📌 Изучим особенности инкапсуляции в языке
- 📌 Узнаем о наследовании и механизме разрешения множественного наследования.
- 📌 Разберёмся с полиморфизмом объектов.

Основы ООП в Python

Классы

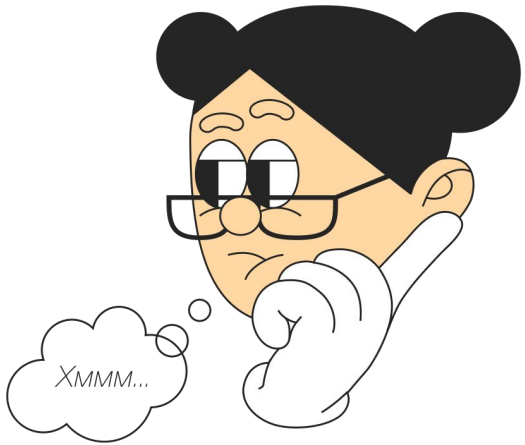
Класс — универсальный, комплексный тип данных, состоящий из тематически единого набора «полей» (переменных более элементарных типов) и «методов» (функций для работы с этими полями), то есть он является моделью информационной сущности с внутренним и внешним интерфейсами для оперирования своим содержимым.



Экземпляры класса

Объект созданный в результате вызова класса называется его экземпляром.

```
p1 = Person()
```

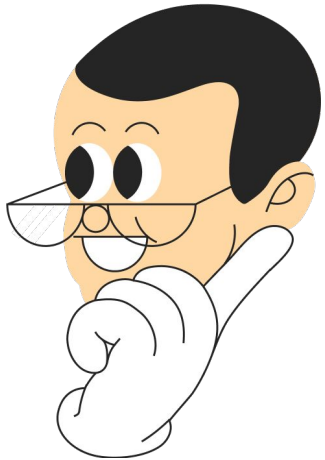


Атрибуты класса и экземпляров

Переменная внутри класса называется атрибутом.
Также можно встретить термины свойство класса или поля класса.

- class Person:
  max_up = 3

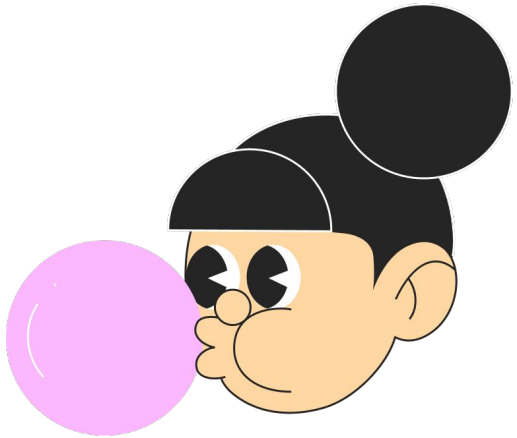
  класс Person имеет атрибут max_up



Конструктор экземпляра

Магический метод __init__ срабатывает при каждом создании экземпляра

```
class Person:
    ...

    def __init__(self):
        ...
```



Параметр self и передача аргументов в экземпляр

Первый параметр функции всегда self — указатель экземпляру на самого себя

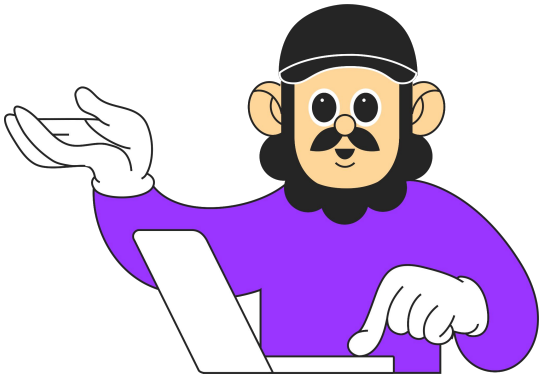
- Первый параметр функции всегда self — указатель экземпляру на самого себя
- Параметры метода __init__ попадают в экземпляр при создании
- Обращение к атрибутам экземпляра происходит через self.name

Методы класса

Функция внутри класса называется методом

- **instance.method()**
  Для доступа к методу экземпляр использует точечную нотацию

- **instance.method(*args, **kwargs)**
  Метод может принимать аргументы как и функция

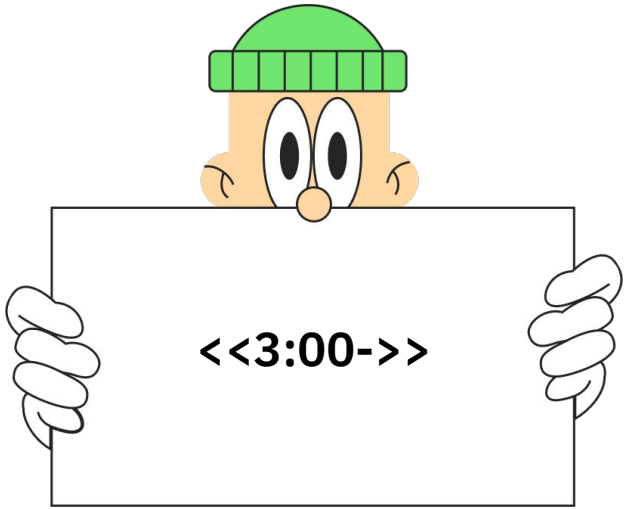


Перед вами несколько строк кода.
Напишите в чат что они вернут
не запуская программу.

У вас 3 минуты.

Основы ООП

```
class User:
    count = []

    def __init__(self, name, phone):
        self.name = name
        self.phone = phone

u1 = User('One', '123-45-67')
u2 = User('NoOne', '76-54-321')
u1.count.append(42)
u1.count.append(73)
u2.counter = 256
u2.count.append(u2.counter)
u2.count.append(u1.count[-1])

print(f'{u1.name = }, {u1.phone = }, {u1.count = }')
print(f'{u2.name = }, {u2.phone = }, {u2.count = }')
```



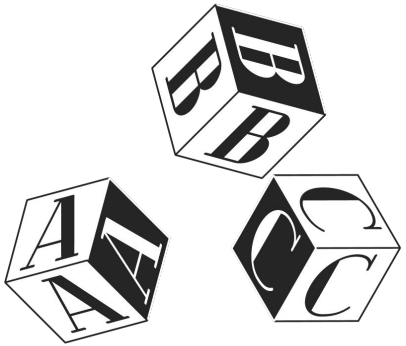
Инкапсуляция

Модификаторы доступа

💡 **public**
публичный доступ, т.е. возможность обратиться к свойству или методу из любого другого класса и экземпляра

💡 **protected**
защищённый доступ, позволяющий обращаться к свойствам и методам из класса и из классов наследников

💡 **private**
приватный доступ, т.е. отсутсвие возмодности обратиться к свойству или методы из другого класса или экземпляра



Инкапсуляция

Символ подчёркивания определяет уровень инкапсуляции

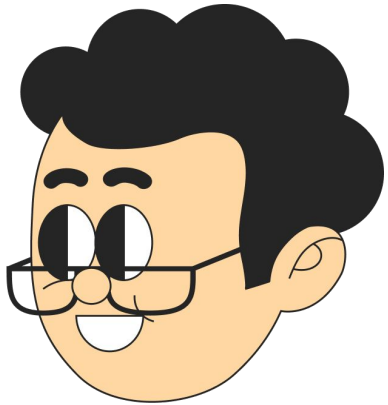
- self._name
  защищённый атрибут (свойство, поле)
- def _method(self):
  защищённый метод
- self.__name
  приватный атрибут (свойство, поле)
- def __method(self):
  приватный метод

Доступ к приватным переменным

Внутренний механизм Python модифицирует приватные имена.

- было
```
class MyClass:
    ...
    self.__name
```

- стало
```
z.MyClass()
z._MyClass__name
```

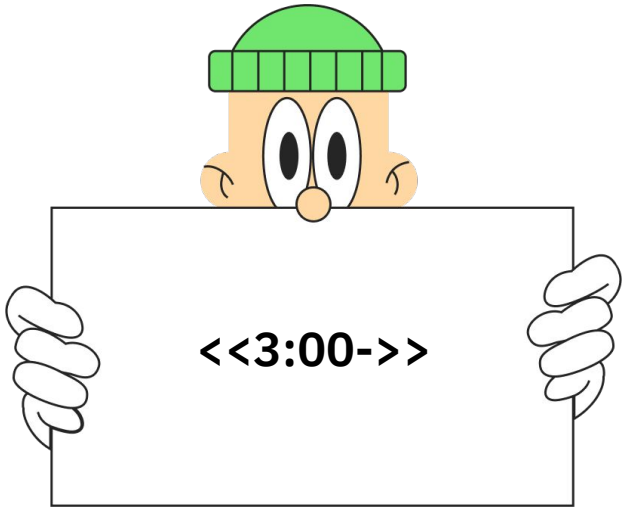


Перед вами несколько строк кода.
Какие ошибки и недочёты есть в коде.

У вас 3 минуты.

Инкапсуляция

```
class User:

    def __init__(self, name, phone, password):
        self.__name__ = name
        self._phone = phone
        self.__password = password


u1 = User('One', '123-45-67', 'qwerty')

print(f'{u1.__name__ = }, {u1._phone = },
{u1._User__password = }')
```



Наследование

Наследование

Все классы Python наследуются от родительского класса

- пишем так

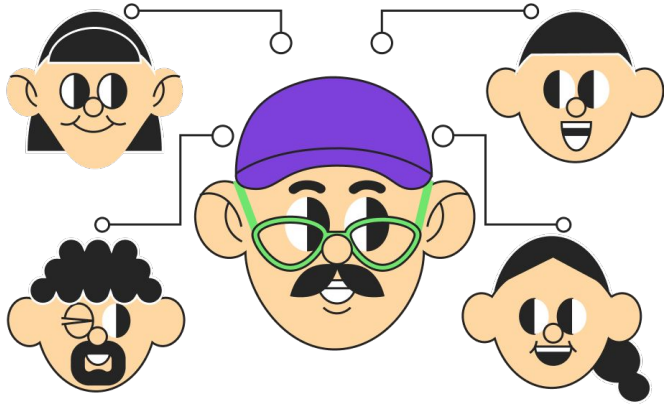
```
class Person:
    pass
```

- подразумеваем это

```
class Person(object):
    pass
```

Переопределение методов

Одноимённые методы дочернего класса переопределяют методы родителя



Множественное наследование

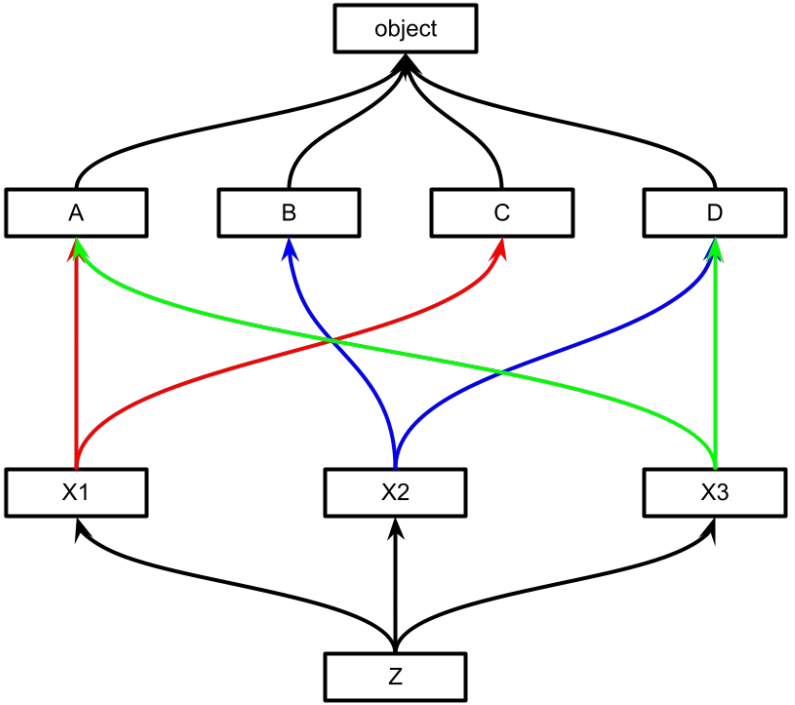
Любой класс может иметь несколько родительских классов

```
class Child(Parent1, Parent2, Parent3):
    ...
```

MRO

MRO — method resolution order переводится как «порядок разрешения методов»

- ClassName.mro()



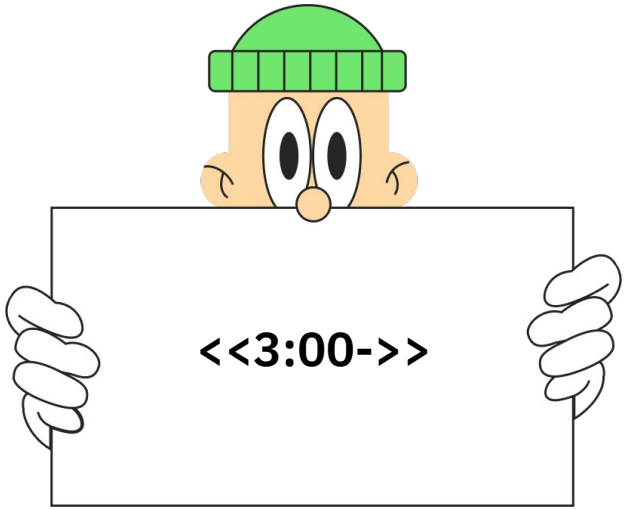
Перед вами несколько строк кода.
Напишите в чат что они вернут
не запуская программу.

У вас 3 минуты.

Наследование

```
class A:
    name = 'A'
    def call(self):
        print(f'I am {self.name}')


class B:
    name = 'B'
    def call(self):
        print(f'I am {self.name}')


class C:
    name = 'C'
    def call(self):
        print(f'I am {self.name}')
```

```
class D(C, A):
    pass


class E(D, B):
    pass


e = E()
e.call()
```

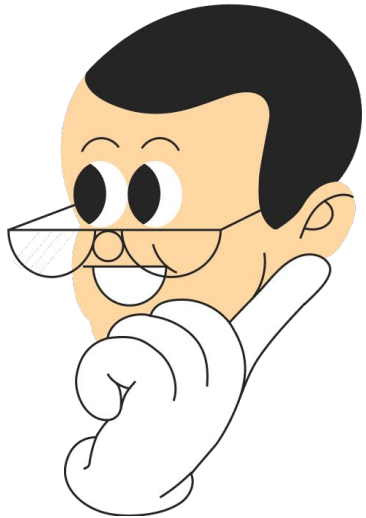


Полиморфизм

Полиморфизм

Полиморфизм — свойство системы, позволяющее использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта.

```
>>> 42 + 73
115
>>> ‘42’ + ‘73’
'4273'
```



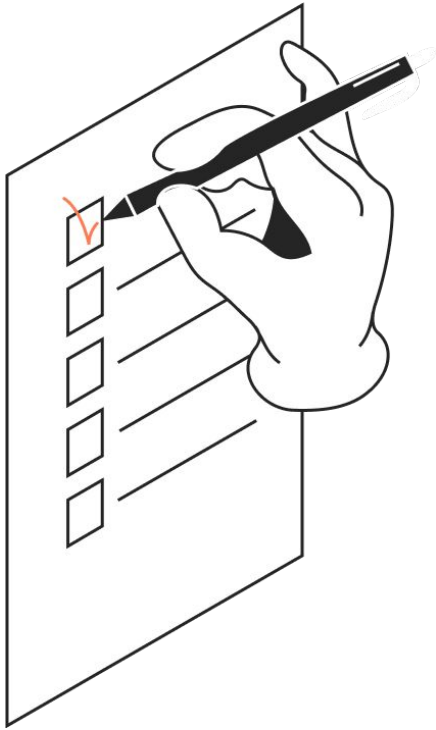
Итоги занятия

На этой лекции мы

- Разобрались с объектно-ориентированным программированием в Python.
- Изучили особенности инкапсуляции в языке
- Узнали о наследовании и механизме разрешения множественного наследования.
- Разобрались с полиморфизмом объектов.

Задание

Возьмите 1-3 задачи из прошлых занятий
и попробуйте перенести переменные и функции
в класс. Убедитесь, что созданный вами класс
позволяет верно решать поставленные задачи.



Спасибо за внимание