

DIRTY PYTHON 1.0

STONE 13TH PRESENTS

ЛЕКЦИЯ 10



DIRTY
PYTHON

ЧТО У НАС СЕГОДНЯ?

Объекты и классы

Атрибуты и методы

Пара слов о дандер-методах

Напишем парочку своих классов и
связем их между собой

Сделаем телефонную книгу на классах

У тебя есть объект?

Лучше, у меня есть класс
объекта.

meme-arsenal.ru



ПАРУ СЛОВ ОБ ОБЪЕКТАХ, КЛАССАХ И ООП



- Объектно-ориентированное программирование – это целое направление, парадигма, которая базируется на использовании объектов и создании классов, но вовсе этим не ограничивается.
- Объекты и классы вполне себе используются и в функциональном программировании – это всего лишь инструмент для решения задач
- Сегодня мы рассмотрим только самую верхушку айсберга



ЧТО ЖЕ ТАКОЕ КЛАСС И ОБЪЕКТ?



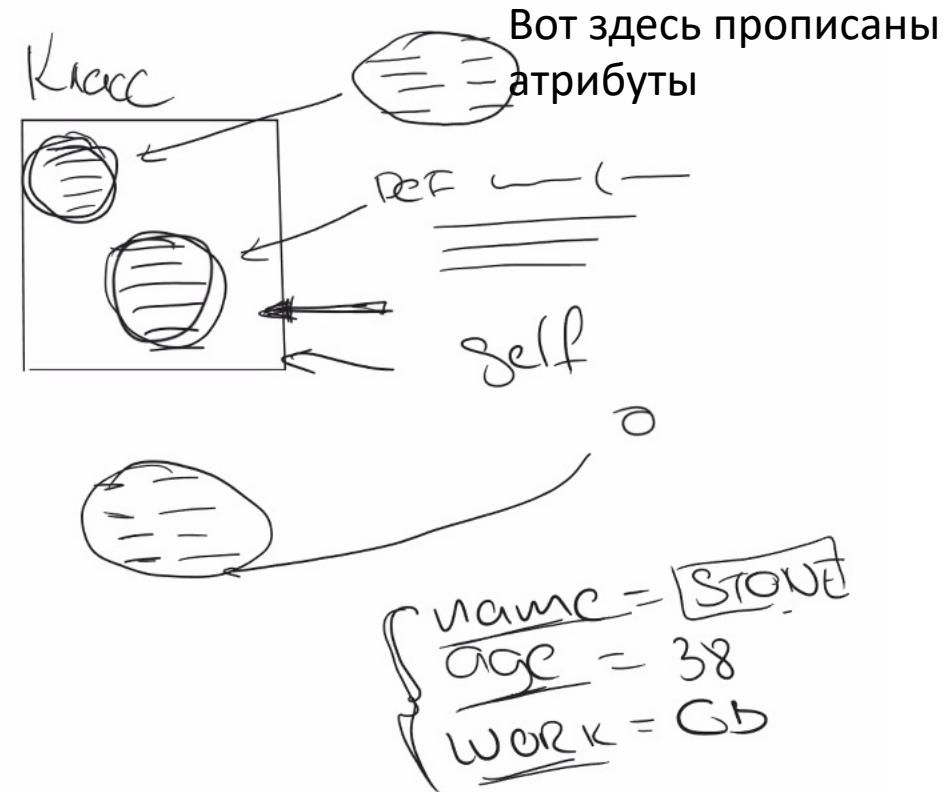
Класс – это некоторая сущность, воплощением которой являются объекты.

А если проще давайте приведем аналогию: чертеж табуретки и сама табуретка. Так вот чертеж – это класс: на него нельзя сесть, но при его помощи можно создать сколько угодно табуреток, а вот табуретка – это физическое воплощение чертежа, то что будет использоваться по назначению, то есть объект



АТРИБУТЫ

- Поля, они же атрибуты/ состояния – прописываются для всего класса, например при помощи `__init__`, где будут перечислены свойства присущие всем объектам данного класса (это могут быть любые типы данных, и даже другие классы). Например, если мы будем писать класс Human, то можно указать такие атрибуты как name, age, sex и т.д. И когда будут создаваться объекты данного класса – эти атрибуты будут принимать конкретные значения. Условно, можно сказать что это похоже на словарь: ключи и значения.

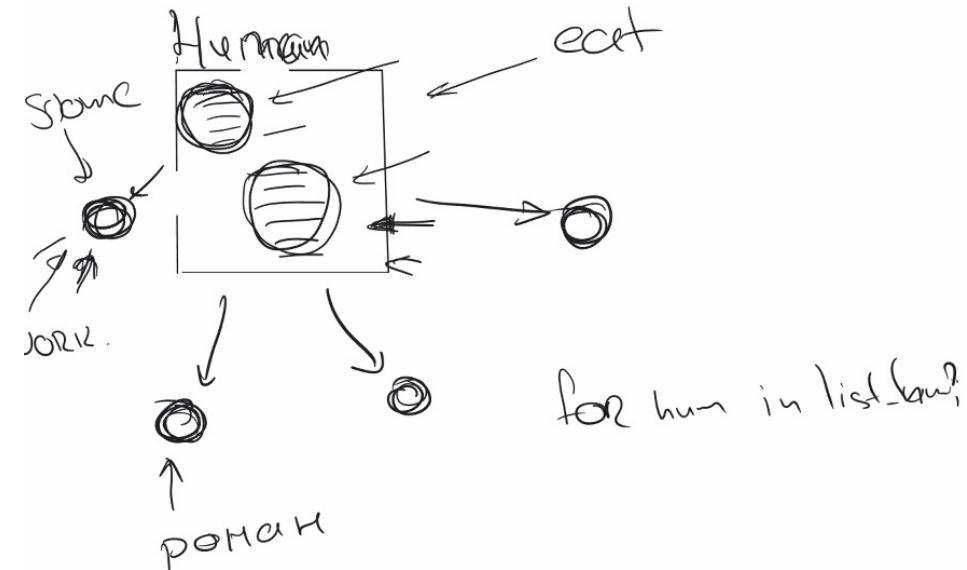


Вот здесь показано, чему они равны у конкретного объекта

МЕТОДЫ



- Это встроенные функции класса, которые работают только со своими объектами и задает их поведение – то, что может делать наш объект.
- Отличие от функции в том, что они находятся внутри класса и принимает сам объект в качестве атрибута (имеуно self)
- При этом если метод прописан внутри класса, то он будет работать для всех! объектов данного класса.
- Если у объекта есть какие-то свои собственные методы, которые не работают для других объектов класса, то что-то пошло не так...



НАСЛЕДОВАНИЕ



- Буквально пару слов про наследование
- Если мы будем на основе нашего класса создавать другой класс, то потомок унаследует все атрибуты и методы родителя, но при этом обзаведется своими собственными. А вот наоборот это не работает.
- Например, у любого класса, который вы будете создавать можно вызвать дандер-методы (названия этих методов окаймлены двойным подчеркиванием, их правильнее очень много!) – все потому, что любой создаваемый класс является наследником общего класса `object`, в котором защищены эти методы
- Но пока глубже не пойдем)



```
7 def ___
8     __rtruediv__(self, ...
9     __rxor__(self, othe...
10    __set__(self, insta...
11    __set_name__(self, ...
12    __setattr__(self, k...
13    __setitem__(self, k... :elf.
14    __setslice__(self, ...
15    __setstate__(self, ...
16    __sizeof__(self) pr...
17    __sub__(self, other)
18    __subclasscheck__(s...
```

ЖИЗНЬ ОБЪЕКТА



- Изначально у всех создаваемых объектов – одинаковые атрибуты (но могут быть с разным содержимым)
- Но ничто не мешает наделить вам отдельный объект каким-то новым свойством, которого не будет у другого объекта в этом классе – это уже приобретенное свойство.
- Можно, конечно, новое свойство прописать в класс – тогда у всех объектов оно тоже будет. Но менять класс в процессе работы программы – вроде и можно, но такое себе, не лучшая практика.

ДАВАЙТЕ ПОСМОТРИМ КОД



```
1 class Human:  
2     def __init__(self, name: str, age: int, work: str = 'GB'):  
3         self.name = name  
4         self.age = age  
5         self.work = work  
6
```

- Чтобы создать класс нам нужно написать класс и название. Можно написать и скобочки, но в них обычно указывают от кого класс наследуется – если он у нас сирота, можно поставить круглые скобки, но лучше не надо
- Дальше пишем функцию `__init__` через которую задаем атрибуты: этот метод срабатывает при создании объекта: прописываем атрибут `self`, название атрибута и его тип, можно задать значение по умолчанию (все как с атрибутами функций)
- Затем прописываем все через `self`



- self – это и есть сам объект: прописывая self.<атрибут/метод> мы вызываем этот метод или свойство у конкретного объекта, к которому применяем
 - Давайте например, создадим объект – это очень легко нужно дать ему название и присвоить ему результат вызова класса с указанием необходимых атрибутов (тех, что у нас прописаны в функции __init__):

```
stone = Human('STONE', 38)
```

Кстати, как и с при вызове функций, если у нас значение указано по умолчанию, то можно не указывать этот аргумент

- Теперь если мы обратимся `stone.name`, то получим именно то, что записано в этом атрибуте именно у объекта `stone`, в данном случае `STONE` – и все благодаря `self`

НАПИШЕМ МЕТОД ДЛЯ КЛАССА



- Для этого нам нужно будет использовать `def <название метода>(self)` – указываем, что метод будет применяться к объекту класса, хотя можно передавать и другие атрибуты
- Если нам внутри функции нужно получить какое-то свойство объекта – опять же делаем это через `self`
- Вызывая метод для объекта не забываете использовать скобочки

```
Scratches and Consoles
```

```
1 usage
2
3
4
5
6
7
8     def greetings(self):
9         return f'Меня зовут {self.name}'
10
11
12 stone = Human('STONE', 38)
13
14 print(stone.greetings())
15
16
```

D:\WorkShopPy\.my_venv\Scripts\python.exe D:\WorkShopPy\DPBASIC_1\main.py

Меня зовут STONE

Process finished with exit code 0

СОЗДАДИМ ПРИОБРЕТЕНОЕ СВОЙСТВО



- Это опять таки легко: пишем объект, через точку название нового свойства и присваиваем значение.
- Все, теперь у объекта stone есть атрибут girlfriend, который тоже можно вызвать, а у объекта roman - нет.
- Такое бывает, но это делается редко: нам нужно работать по шаблону, чтобы все методы работали одинаково
- Если у нас кто-то такой особенной – лучше сделать наследуемый класс.

```
11 stone = Human('STONE', 38)
12 roman = Human('Роман', 18)
13 print(stone.greetings())
14 print(roman.greetings())
15
16 stone.girlfriend = 'Anna'
17
18 print(stone.girlfriend)
19 print(roman.girlfriend)
20
```

```
Run main.py
AttributeError: 'Human' object has no attribute 'girlfriend'
Меня зовут STONE
Меня зовут Роман
Anna
```

ЗАДАЕМ АТРИБУТЫ В INIT

Мы можем создавать какие-то свойства для объектов относительно других их свойств, а не прописывать через атрибуты.

Например зададим свойство junior – если возраст меньше 18 – True, иначе False

```
class Human:  
    def __init__(self, name: str, age: int, work: str = 'GB'):  
        self.name = name  
        self.age = age  
        self.work = work  
        self.junior = True if age < 18 else False
```

```
11  
12 stone = Human('STONE', 38)  
13 roman = Human('Роман', 14)  
14 print(stone.junior)  
15 print(roman.junior)  
16  
17 |  
  
Run main.py  
u:\workshop\my_venv\Scripts\python.exe u:\workshop\UPBAS10.  
False  
True  
  
Process finished with exit code 0
```

ИСПОЛЬЗУЕМ ДРУГИЕ КЛАССЫ В МЕТОДАХ

Создадим класс Car со своими атрибутами

Теперь можем сделать метод, чтобы поменять атрибут car у объекта класса Human (по умолчанию None), и передать туда объект класса car.

Теперь в метод buy_car нужно передать объект класса car, а все другие типы (например, если вы присвоите строку) не сработают

Более того, можно у класса Car прописать метод drive, а у класса Human метод drive_car , который будет выполняться через вызов у объекта car, который связан с объектом human ...

И это еще не самый сложный способ взаимодействия между классами

```
1 usage
2 class Car:
3     def __init__(self, name: str, year: int, color: str):
4         self.name = name
5         self.year = year
6         self.color = color
7
8     def drive(self):
9         return 'Бrrrrrrr-р-р-р-р-р-р'
```

```
1 usage
2 def buy_car(self, car: Car):
3     self.car = car
4
5     def drive(self):
6         return self.car.drive()
```

ЕЩЕ НЕМНОГО О КЛАССАХ В КЛАССАХ

По умолчанию у всех объектов Human атрибут car – None

Через метод `by_car` мы добавили объекту stone в качестве значения объект класса Car, указав в скобках соответствующие атрибуты

И вот теперь можно у stone вызвать метод `drive`

А у Романа все еще нет машины, пусть это и розовая мазда 1976 года....

```
29     stone = Human('STONE', 38)
30     roman = Human('Роман', 14)
31     print(stone.car)
32     print(roman.car)
33     stone.buy_car(Car('Mazda', 1976, 'pink'))
34     print(stone.drive())
35     print(roman.car)
36
```

```
Run main x
U:\WorkShopPy\my_venv\Scripts\python.exe U:\WorkShopPy\УРОВНИ_1\main.py
None
None
Брррррр-р-р-р-р-р!
None
```



МУДРОСТЬ ОТ СТОУНА



TYPE HINTING ОЧЕНЬ ВАЖНО, КОГДА
НАЧИНАЮТСЯ КАСТОМНЫЕ КЛАССЫ БЕЗ НЕГО
ПРОСТО НИКАК!

НЕМНОГО О ДАНДЕР-МЕТОДАХ

Если вы напишете просто `print(stone)` то вам выведет в консоль, где лежит этот объект, а не его атрибуты как вы могли бы подумать...

То же кстати, если вы захотите сделать `print(int)` или `str`. Кстати, если через `enter` провалится в `int` или `str`, то вам вывалится описание этих классов со всеми их методами...

К чему это все? А к тому, что есть дандер-метод `__str__` который позволяет определить, что выводить при печати объекта

```
32 print(stone)
33
D:\WorkShopPy\.my_venv\Scripts\python.exe D:\WorkShopPy\D
<__main__.Human object at 0x000002376EBBD0>

4935 ❶ class str(object):
4936      """
4937          str(object='') -> str
4938          str(bytes_or_buffer[, encoding[, errors]]) -> str
4939
4940          Create a new string object from the given object. If encoding or
4941          errors is specified, then the object must expose a data buffer
4942          that will be decoded using the given encoding and error handler.
4943          Otherwise, returns the result of object.__str__(). (if defined)

28 ❹
29      def __str__(self):
30          return f'Это {self.name}, ему {self.age} и она работает в {self.work}' + (f'У него есть машина'
31                                          f' {self.car}' if self.car else f'У него нет машины')
32
33      stone = Human('STONE', 38)
34      roman = Human('Роман', 14)
35
36      print(stone)
```

НЕМНОГО О ЛАНДЕР- МЕТОДАХ



```
main.py
  2 usages
External Libraries
Scratches and Consoles
  1 usage (1 dynamic)
  class Car:
    def __init__(self, name: str, year: int, color: str):
      self.name = name
      self.year = year
      self.color = color
    def __str__(self):
      return f'{self.name}, {self.year} года выпуска. {self.color}'
  1 usage (1 dynamic)
  class Human:
    def __init__(self, name: str, age: int, work: str = 'GB'):
      self.name = name
      self.age = age
      self.work = work
      self.junior = True if age < 18 else False
      self.car = None
    def greetings(self):
      return f'Меня зовут {self.name}'
  1 usage
Car > drive()

U:\WorkShopPy\.my_venv\Scripts\python.exe U:\WorkShopPy\DPBASIC_1\main.py
Это STONE, ему 38 и он работает в GB. У него нет машины
Это STONE, ему 38 и он работает в GB. У него есть машина Mazda, 1970 года выпуска. черненький
```

Точно так же мы можем переопределить метод `__str__` и для класса `car` – и тогда все вообще красиво

ПИШЕМ ТЕЛЕФОННЫЙ СПРАВОЧНИК НА КЛАССАХ



В телефонной книге у нас будут контакты – значит, нужен класс контактов

Да и саму телефонную книгу тоже нужно посадить на класс и прописать все методы:

Открыть, показать контакты, добавить контакт, найти контакт, удалить контакт, изменить контакт...

```
3  class Contact:
4      def __init__(self, name: str, phone: str, comment: str):
5          self.name = name
6          self.phone = phone
7          self.comment = comment
8
9      def __str__(self):
10         return f'{self.name: <20} | {self.phone: <20} | {self.comment: <20}'
```

ТЕЛЕФОННАЯ КНИГА



```
class PhoneBook:
    def __init__(self, path: str = 'pb.txt'):
        self.path = path
        self.contact: list[Contact] = []

    usage

    def open(self):
        with open(self.path, 'r', encoding='UTF-8') as file:
            data = file.readlines()
        self.contact = [Contact(*list(map(lambda x: x.strip(), contact.split(':')))) for contact in data]

    def __str__(self):
        return '\n'.join([contact.__str__() for contact in self.contact]) if self.contact else 'Телефонная книга пуста'

    usage
```

В классе телефонная книга укажем такие параметры как путь (зададим по умолчанию), и список контактов как список объектов класса Contact

Дальше пишем все нужные методы – открыть книгу, распечатать ее при помощи дандер метода `__str__`

Ну и все остальные тоже напишем

ТЕЛЕФОННАЯ КНИГА



```
def add(self, data: Contact | list[str] | str):  
    if isinstance(data, str):  
        self.contact.append(Contact(*list(map(lambda x: x.strip(), data.split(':')))))  
    elif isinstance(data, list):  
        self.contact.append(Contact(*data))  
    elif isinstance(data, Contact):  
        self.contact.append(data)  
    else:  
        raise "Ошибка типа данных"
```

Универсальный метод добавления контактов – через строчку, которую рассплитуем и стрипнем, если передатут список то и его добавим, или напрямую добавлением объекта класса Contact

А если попробуют запихнуть что-то другое отработаем ошибку

ТЕЛЕФОННАЯ КНИГА



```
DPBASIC_1 D:\WorkShopPy\DPBASIC_1\main.py
39     phonebook.open()
40
41
42     phonebook.add('Вера:348757639:Даёт')
43     phonebook.add(['Валера', '3463463', 'ОПППППАСНЫЙ тип'])
44     cont = Contact('Игорь', '253235', 'Зажихай')
45     phonebook.add(cont)
46     print(phonebook)
47
48
49
```

Run main

Имя	Номер телефона	Тип
Панфилов Кирилл	89694512021	Семинари 68
Рокан Ленихов	8990988998	Молодец
Андрей Белаяв	9900	Друг Стоуна
Диана Мороз	80-99-080	Пиздатые конспекты
Вера	348757639	Даёт
Валера	3463463	ОПППППАСНЫЙ тип
Игорь	253235	Зажихай

Process finished with exit code 0

И вот теперь открываем книгу, добавляем в нее контакты разными способами и просто отправляем через `print()`
Ну разве не красота, а?

И ЧТО ДАЛЬШЕ?

- А ТЕПЕРЬ СОБЕРИТЕ В КУЧУ ВСЕ ВАШИ ЗНАНИЯ И НАПИШИТЕ ИГРУ КРЕСТИКИ-НОЛИКИ:
 - ✓ КОНСОЛЬНЫЙ ВАРИАНТ (ИЛИ ДЛЯ БОТА, ИЛИ В TKINTER)
 - ✓ ПРОТИВ ЧЕЛОВЕКА ИЛИ ПРОТИВ КОМПА
 - ✓ ИЛИ ПРОТИВ КОМПА С ЛОГИКОЙ (ИИ ТАК СКАЗАТЬ)))
- ЕСЛИ И ЭТОГО МАЛО - НАПИШИТЕ НА КЛАССАХ ЧТО-ТО ВРОДЕ КОФЕЙНОГО АВТОМАТА С НАПИТКАМИ ИЛИ БАНКОМАТА С КАРТАМИ



D1RTY
РУТНОН

ИТОГИ КУРСА

- ПОЗНАКОМИЛИСЬ С ТИПАМИ ДАННЫХ, ВЕТВЛЕНИЯМИ, ЦИКЛАМИ, ФУНКЦИЯМИ, ФУНКЦИЯМИ ВЫСШЕГО ПОРЯДКА, РАБОТОЙ С ФАЙЛАМИ, СИТАКСИЧЕСКИМ САХАРОМ, ПАРСИНГОМ, МОДУЛЬНОСТЬЮ И АРХИТЕКТУРОЙ, ОБЪЕКТАМИ И КЛАССАМИ
- НО ЭТО ТОЛЬКО САМОЕ НАЧАЛО ПУТИ! НАДЕЕМСЯ, ТЕПЕРЬ ВАМ БУДЕТ ЛЕГЧЕ ЕГО ОСИЛИТЬ)
- УСПЕХОВ ВАМ!
- КОДЬТЕ БОЛЬШЕ, КОДЬТЕ ЧАЩЕ, КОДЬТЕ ЛУЧШЕ!