Python
Основы ООП

с использованием иллюстраций и материалов, подготовленных Кондюриной А.А.

Вспомнить всё

Что такое информатика?

- Наука об информации

Что такое информация?

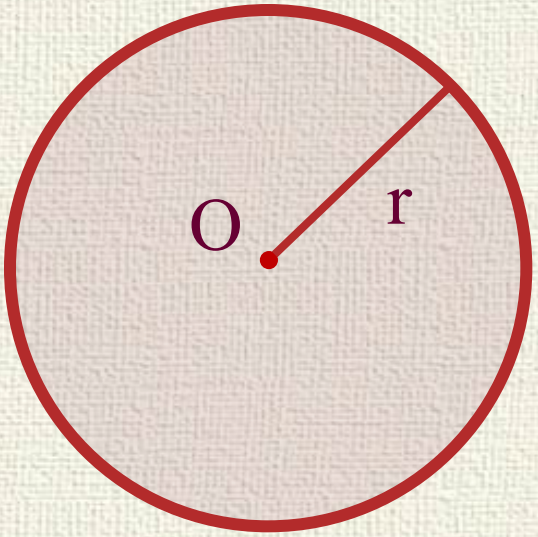
- Данные об окружающем мире

Что такое программа?

- Набор команд (алгоритм) для компьютера, производящий обмен данными между человеком и компьютером.

Что такое окружность?

Для человека:



Для компьютера:

x=5

y=5

r=7

Как хранятся данные в программе?

- переменные

Как осуществляются действия с данными?

- функции

Связанные данные

Вариант 1

x=5

y=5

r=7

Вариант 2

circle = {‘x’:5, ’y’:5, ’r’:7}

Сложные типы данных позволяют хранить не только сами данные, но и связь между ними. Это позволяет взаимодействовать, как с частью данных, так и со всеми данными целиком.

dict_of_circles = [{‘x’:randint(1,100), ’y’:randint(1,100), ’r’:randint(1,100)} for i in range(10)]

Глупые кожаные мешки

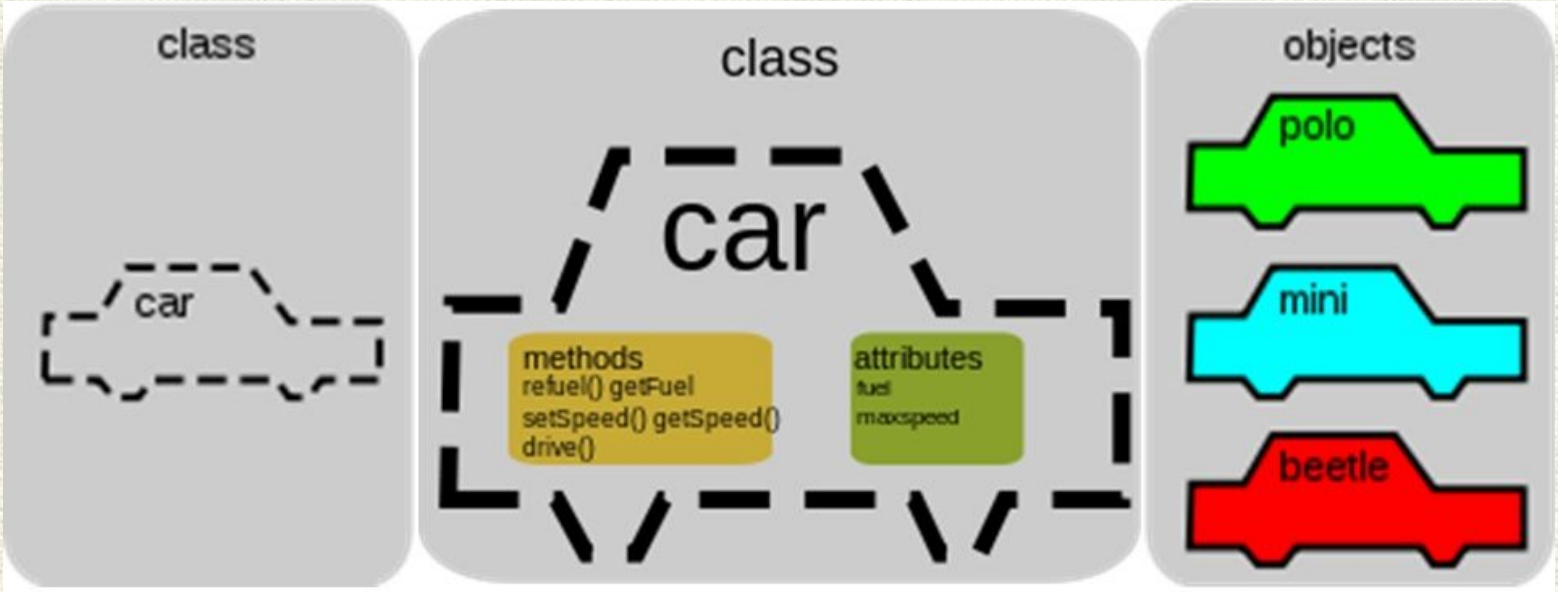
Что делает человек при решении сложных задач?

Одушевление предметов…

«А что если внутри описания алгоритма работы функции описать работу ещё одной функции…»

И что же получилось?

Объектно-ориентированное программирование (ООП) – парадигма программирования, в которой программа представляет собой совокупность взаимодействующих объектов.



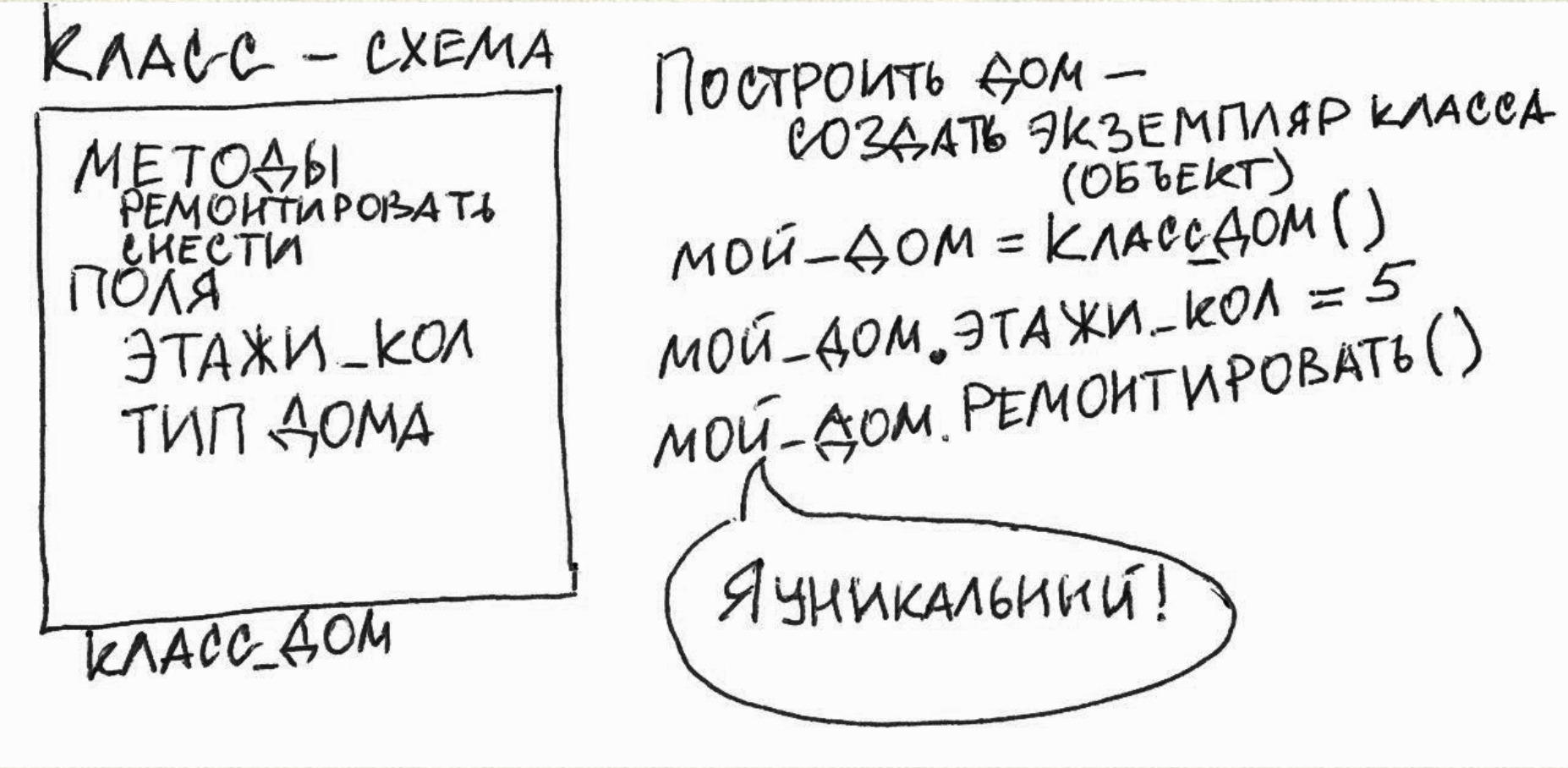
Парадигма – набор концепций, шаблонов мышления. ООП – это своего рода «философия» написания кода.

Классы и объекты

Объект – конкретный экземпляр класса. Способен хранить данные (атрибуты) и выполнять действия (методы). Обладает состоянием и поведением.

Класс – тип данных, описывающий модель создания объектов. Содержит структуру хранения данных (набор полей и их начальное состояние), и алгоритмы (методы) для работы с объектами.

Классы и объекты



И как их делать?

```
class Имя_класса:
    # определение атрибутов и методов класса

объект = Имя_класса() # создание объекта класса
```

```
class Circle:
    def __init__(self):      #метод
        self.x=int(input()) #атрибут
        self.y=int(input()) #атрибут
        self.r=int(input()) #атрибут
    def resize(self, size=2):      #метод
        self.r+=size

circle_1 = Circle()  # создание объекта класса
print(circle_1.x, circle_1.y, circle_1.r)
circle_1.resize(5)
print(circle_1.r)
```

 # Имена классов принято писать с заглавной буквы вот так: MyClass

Что внутри класса?

Атрибут – значение (поле, свойство, параметр), характеристика объекта или класса.

Метод – функция, действие, выполняемое с объектом или над ним.

Часто при создании объекта необходимо сразу задать значения всем (или части) его атрибутов. Для этого внутри класса описывается специальный метод.

Конструктор (__init__()) — это метод, который вызывается при создании экземпляра класса. Необходим для задания значения атрибутам экземпляра при его создании.

```
class Point:
# конструктор класса с параметрами
   def __init__(self, x, y, z):
self.coord = (x, y, z)

p = Point(0.0, 1.0, 0.0)  # вызов конструктора
```

А кто это сделал?

Если много объектов, то как запомнить, к какому объекту привязано значение?

self — это первый формальный параметр метода класса, который содержит ссылку на тот объект, который вызывает метод.

```
circle_1 = Circle()
print(circle_1.x, circle_1.y, circle_1.r)

circle_2 = Circle()
circle_2.x = 100
circle_2.y = 200
print(circle_2.x, circle_2.y, circle_2.r)
```

Атрибуты

Атрибут экземпляра — значение, которое определено для каждого экземпляра отдельно. Описываются в конструкторе или других методах класса.

```
class Dolmatin:
    def __init__(self):
        self.collar=int(input())
    def set_name(self, name):
        self.name=name

puppy=Dolmatin()
puppy.set_name("Tuzik")
```

Также можно создать атрибут экземпляра не описанный в классе. Для этого нужно обратиться к атрибуту в основной программе.

```
puppy.training = True
```

Атрибуты

Атрибут класса — значение, доступное для всех экземпляров данного класса. Определяется внутри класса, но вне любых методов.

```
class Dolmatin:
    color='spotted'
```

Обращение к атрибуту класса

1. Через имя класса:

```
puppy=Dolmatin()
print(Dolmatin.color)
```

2. Через объект класса, если у объекта нет атрибута с таким же именем:

```
puppy=Dolmatin()
print(puppy.color)
```

Атрибуты и методы

```
class Negr:
    # атрибуты класса.
    color = ‘Black’
    count = 0

    # конструктор класса. вызывается при создании экземпляра
    def __init__(self, name, job):
    # атрибуты экземпляра, их значения доступны только объекту
        self.name = name
        self.job = job
        # увеличиваем значение переменной класса
        Negr.count += 1

    def info(self):
        print('Имя: ', self.name)
        print( ‘Профессия: ', self.job)
        print( 'Всего: ', Negr.count)
```

Атрибуты и методы

```
negr_1 = Negr(‘Lui', ‘jazzman')
negr_1.info()  # Всего: 1

negr_2 = Negr(‘Michael', ‘moonwalker')
negr_2.info()  # Всего: 2
```

Атрибут класса color общий для всех экземпляров класса.

```
negr_2.color = ‘White'

print(negr_1.color)
print(negr_2.color)
print(Negr.color)

Negr.color = …
```

Передача данных

```
class Circle:
    def __init__(self, values):
        self.x = values[0]
        self.y = values[1]
        self.r = values[2]
    def info(self):
        print('x coord=', self_1.x)
        print('y coord=', self_1.y)
        print('r value=', self_1.r)

circle_1 = Circle([2,2,6])
circle_1.info()

circle_2 = Circle([int(input()), int(input()), int(input())])
circle_2.info()


list_of_circles = [Circle([randint(1,10), randint(1,10), 5]) for i in range(5)]
for i in list_of_circles:
        i.info()
```

Передача данных

```
def resize(self, size=2):
    self.r+=size

class Circle:
    newsize = resize
    def __init__(self, values):
        self.x = values[0]
        self.y = values[1]
        self.r = values[2]
    def info(self):
        print('x coord=', self_1.x)
        print('y coord=', self_1.y)
        print('r value=', self_1.r)

circle_1 = Circle([2,2,6])
circle_1.info()
circle_1.newsize()
circle_1.info()
```

Т.к. в Python функции являются объектами и их можно хранить, как значения, то методы класса можно определять вне тела класса в виде функции и передавать внутрь тела в виде атрибута класса.

И у таких функций должен быть обязательный параметр self!

Можно? Нужно!

```
class Negr:
  …

   def kick(self, other):
       self.power = 100
       other.sadness = True
       print( ‘Mister', self.name, ‘kick Mister’, other.name )

negr_1 = Negr(‘John’, ‘bodyguard’)
negr_2 = Negr(‘Valentain’, ‘paparazzi’)
negr_1.kick(negr_2)
# ‘Mister John kick Mister Valentain’
```

Данные других объектов также могут использоваться внутри методов исходного объекта.

И зачем оно надо?

Преимущества:

1) Модульность

ООП позволяет сделать код более структурированным и независимым. Упрощает работу нескольких разработчиков.

2) Гибкость

ООП-код легко развивать, дополнять и изменять. Взаимодействие с объектами, а не логикой упрощает понимание кода.

3) Экономия времени

Можно не писать один и тот же код много раз и использовать заново в новых проектах.

4) Безопасность

При правильной организации кода, данные защищены от доступа извне на уровне самого языка программирования.

И зачем оно надо?

Недостатки:

1) Высокий порог входа

Для понимания и использования ООП сначала нужно освоить процедурное программирование.

2) Ресурсоемкость

ООП-код требует больше ресурсов из-за особенностей доступа к данным, а также хранения и обработки большого количества связанных данных.

3) Большой объем кода

Код, написанный с использованием ООП, обычно длиннее и занимает больше места на диске, чем «процедурный». Это происходит т.к. ООП ориентировано на добавление нового кода, а не на изменение уже существующего.