Объектно-ориентированное программирование. Введение

План

1. Понятие ООП
2. Развитие идеологии ООП
3. Принципы ООП
4. Класс, объект в ООП
5. Разработчики ООП
6. Языки ООП
7. ООП в Python: создание класса, экземпляра класса, метода
8. Инициализация экземпляра класса
9. Соглашения об именовании

Объектно-ориентированное программирование (ООП) —

методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определённого класса, а классы образуют иерархию наследования

ООП возникло в результате развития идеологии:

- **процедурного программирования**, где данные и подпрограммы (процедуры, функции) их обработки формально не связаны.
- **событийно-ориентированного программирования**
- **компонентного программирования** - дальнейшее развития объектно-ориентированного программирования, при котором часто большое значение имеют понятия события и компонента

Процедурное программирование —

программирование на императивном языке, при котором последовательно выполняемые операторы можно собрать в подпрограммы, то есть более крупные целостные единицы кода, с помощью механизмов самого языка

Событийно-ориентированное программирование

— парадигма программирования, в которой выполнение программы определяется событиями — действиями пользователя (клавиатура, мышь, сенсорный экран), сообщениями других программ и потоков, событиями операционной системы (например, поступлением сетевого пакета)

Компонентно-ориентированное программирование

— парадигма программирования, существенным образом опирающаяся на понятие **компонента** — независимого модуля исходного кода программы, предназначенного для повторного использования и развёртывания и реализующегося в виде множества языковых конструкций (например, «классов» в объектно-ориентированных языках программирования), объединённых по общему признаку и организованных в соответствии с определёнными правилами и ограничениями.

Принципы объектно-ориентированной парадигмы программирования

- Данные структурируются в виде объектов, каждый из которых имеет определенный тип, то есть принадлежит к какому-либо классу.
- Классы – результат формализации решаемой задачи, выделения главных ее аспектов.
- Внутри объекта инкапсулируется логика работы с относящейся к нему информацией.
- Объекты в программе взаимодействуют друг с другом, обмениваются запросами и ответами.
- При этом объекты одного типа сходным образом отвечают на одни и те же запросы.
- Объекты могут организовываться в более сложные структуры, например, включать другие объекты или наследовать от одного или нескольких объектов.

Основные принципы структурирования в случае ООП связаны с различными аспектами базового понимания предметной задачи, которое требуется для оптимального управления соответствующей моделью:

- абстракция для выделения в моделируемом предмете важного для решения конкретной задачи по предмету, в конечном счёте — контекстное понимание предмета, формализуемое в виде класса. Абстрагирование означает выделение значимой информации и исключение из рассмотрения незначимой. В ООП рассматривают лишь абстракцию данных (нередко называя её просто «абстракцией»), подразумевая набор наиболее значимых характеристик объекта, доступных остальной программе.

- инкапсуляция для быстрой и безопасной организации собственно иерархической управляемости: чтобы было достаточно простой команды «что делать», без одновременного уточнения как именно делать, так как это уже другой уровень управления. Инкапсуляция — свойство системы, позволяющее объединить данные и методы, работающие с ними, в классе

Основные принципы структурирования в случае ООП связаны с различными аспектами базового понимания предметной задачи, которое требуется для оптимального управления соответствующей моделью:

- наследование для быстрой и безопасной организации родственных понятий: чтобы было достаточно на каждом иерархическом шаге учитывать только изменения, не дублируя всё остальное, учтённое на предыдущих шагах. Наследование — свойство системы, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствующейся функциональностью. Класс, от которого производится наследование, называется базовым, родительским или суперклассом. Новый класс — потомком, наследником, дочерним или производным классом
- полиморфизм для определения точки, в которой единое управление лучше распараллелить или наоборот — собрать воедино. Полиморфизм — свойство системы, позволяющее использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта.

Класс в ООП

— Описывает модель объекта, его свойства и поведение.

Класс — такой тип данных, который создается для описания сложных объектов.

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 (значениями полей).

В частности, в классах широко используются специальные блоки из одного или чаще двух спаренных методов, отвечающих за элементарные операции с определённым полем (интерфейс присваивания и считывания значения), которые имитируют непосредственный доступ к полю. Эти блоки называются «свойствами» и почти совпадают по конкретному имени со своим полем.

Другим проявлением интерфейсной природы класса является то, что при копировании соответствующей переменной через присваивание копируется только интерфейс, но не сами данные, то есть класс — ссылочный тип данных.

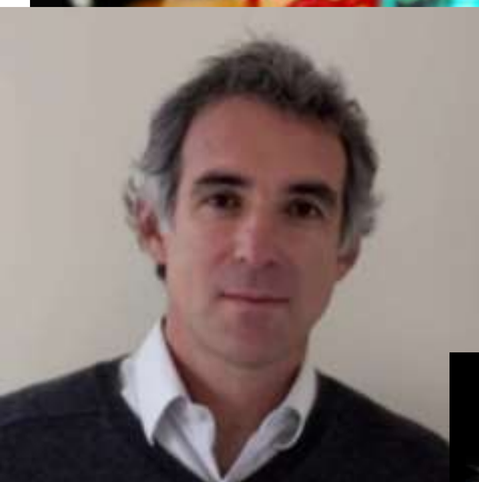
Объект

— некоторая сущность в цифровом пространстве, обладающая определённым состоянием и поведением, имеющая определённые свойства (атрибуты) и операции над ними (методы).

Объект, наряду с понятием класс, является важным понятием объектно-ориентированного подхода. Объекты обладают свойствами наследования, инкапсуляции и полиморфизма

Как правило, при рассмотрении объектов выделяется то, что объекты принадлежат одному или нескольким классам, которые определяют поведение (являются моделью) объекта. Термины «**экземпляр класса**» и «**объект**» взаимозаменяемы. Переменная-объект, относящаяся к заданному классом типу, называется экземпляром этого класса.

При этом в некоторых исполняющих системах класс также может представляться некоторым объектом при выполнении программы посредством динамической идентификации типа данных. Обычно классы разрабатывают таким образом, чтобы обеспечить отвечающие природе объекта и решаемой задаче целостность данных объекта, а также удобный и простой интерфейс. В свою очередь, целостность предметной области объектов и их интерфейсов, а также удобство их проектирования, обеспечивается наследованием.



Разработчики ООП

Лука Карделли и Мартин Абади построили теоретическое обоснование ООП

Алан Кёртис Кэй — американский учёный в области теории вычислительных систем. Один из пионеров в областях объектно-ориентированного программирования и графического интерфейса. Также известен благодаря высказыванию «лучший способ спрогнозировать будущее — изобрести его».

- Разработал язык программирования Smalltalk, где впервые был применён объектно-ориентированный подход.

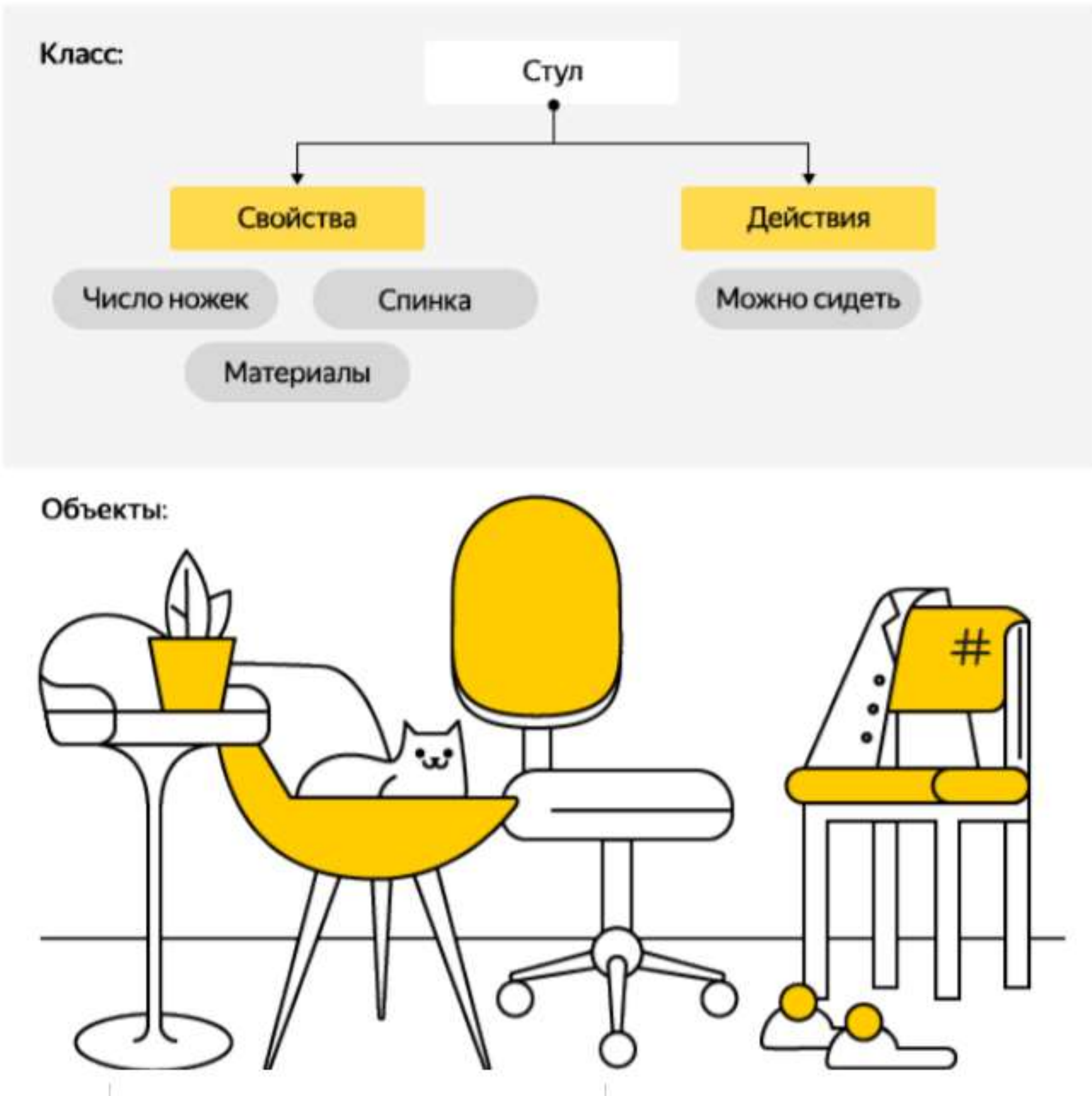
Принципы ООП Алана Кэйя

- Всё является объектом.
- Вычисления осуществляются путём взаимодействия (обмена данными) между объектами, при котором один объект требует, чтобы другой объект выполнил некоторое действие.
- Объекты взаимодействуют, посылая и получая сообщения. Сообщение — это запрос на выполнение действия, дополненный набором аргументов, которые могут понадобиться при выполнении действия.
- Каждый объект имеет независимую память, которая состоит из других объектов.
- Каждый объект является представителем класса, который выражает общие свойства объектов (таких, как целые числа или списки).
- В классе задаётся поведение (функциональность) объекта. Тем самым все объекты, которые являются экземплярами одного класса, могут выполнять одни и те же действия.
- Классы организованы в единую древовидную структуру с общим корнем, называемую иерархией наследования. Память и поведение, связанное с экземплярами определённого класса, автоматически доступны любому классу, расположенному ниже в иерархическом дереве.

Объектно-ориентированный язык программирования (ОО-язык) — язык, построенный на принципах объектно-ориентированного программирования.

- ActionScript
- Ada
- **C#**
- **C++**
- Cyclone
- D
- Delphi
- Dylan
- Eiffel
- F#
- Groovy
- Io
- **Java**
- **JavaScript**
- **JScript .NET**
- **Kotlin**
- Object Pascal
- Objective-C
- Perl
- **PHP**
- PowerBuilder
- **Python**
- Ruby
- Scala
- Simula
- Smalltalk
- Swift
- Vala
- VB.NET
- Visual DataFlex
- Xbase++
- X++

ООП в Python



У каждого объекта есть набор свойств и действия, в которых он может участвовать. Основываясь на этих свойствах (наличие сиденья) и действиях (на стуле можно сидеть), мы классифицируем объекты, то есть относим их к тому или иному классу

- **Класс -** описывает модель объекта, его свойства и поведение. Говоря языком программиста, класс — это такой тип данных, который создается для описания сложных объектов.
- **Объект** - хранит конкретные значения свойств и информацию о принадлежности к классу. Может выполнять методы.
- **Экземпляр** - для краткости вместо «Объект, порождённый классом „Стул“» говорят «экземпляр класса „Стул“».

- **Атрибут** - свойство, присущее объекту. Класс объекта определяет, какие атрибуты есть у объекта. Конкретные значения атрибутов — характеристика уже не класса, а конкретного экземпляра этого класса, то есть объекта.

- **Метод** - действие, которое объект может выполнять над самим собой или другими объектами.

Пример

В игре может быть **класс** *Character (персонаж)*, а его **экземплярами** будут *goodness* или *evil*.

Свойства (**атрибуты**) — это данные, которые связаны с конкретным **объектом**:

- здоровье; очки;
- деньги; сила;
- ловкость; интеллект;
- скорость; координаты.

Поведение объекта определяется с помощью **методов** — специальных блоков кода, которые можно вызывать из разных частей программы.

Например, того же **объекта** *Character* могут быть следующие **методы**:

- идти;
- атаковать;
- говорить;
- подобрать;
- выбросить;
- использовать.

Примеры

```
1, 2, 3, 'abc', [10, 20, 30]  # объекты
int, str, list                # классы
```

```
1, 2, 3        # экземпляры класса int
'abc'          # экземпляр класса str
[10, 20, 30] # экземпляр класса list, в который
               вложены экземпляры класса int
```

Как узнать класс объекта?

Чтобы узнать, к какому классу относится тот или иной объект, можно воспользоваться функцией **type**.

```
type(123)
# => '<class 'int'>'
```

```
type([1, 2, 3])
# => '<class 'list'>'
```

Создание класса

```python
class Fruit:
    pass
```

Определение этого класса состоит из зарезервированного слова **class**, имени класса и пустой инструкции после отступа.

Внутри класса с дополнительным уровнем отступов должны определяться его методы, но сейчас их нет.

Однако хотя бы одна инструкция должна быть, поэтому приходится использовать пустую инструкцию-заглушку **pass**.

Правила PEP 8

Имена классов по стандарту именования PEP-8 должны начинаться **с большой буквы**.

Встроенные классы (int, float, str, list и другие) этому правилу не следуют, однако в вашем коде его лучше придерживаться. Так делает большинство программистов на Python.

Создание экземпляра класса

a = Fruit()

b = Fruit()

Создание атрибутов экземпляра класса

a.name = 'apple'

a.weight = 120

теперь a - это яблоко весом 120 грамм

b.name = 'orange'

b.weight = 150 # b - это апельсин весом 150 грамм

- c = Fruit()
- c.name = 'lemon‘
- c.color = 'yellow'
- print(c.name, c.weight)
- **# Ошибка AttributeError**, нет атрибута weight

Создание метода класса

```python
class Greeter:
    def hello_world(self):
        print("Привет, Мир!")


greet = Greeter()
greet.hello_world() # выведет "Привет, Мир!"
```

При создании собственных методов обратите внимание на два момента:
- Метод должен быть определён внутри класса (добавляется уровень отступов);
- У методов всегда есть хотя бы один аргумент, и первый по счёту аргумент должен называться **self**

Аргумент self

В него передается тот объект, который вызвал этот метод. Поэтому self ещё часто называют **«контекстным объектом»**.

`greet.hello_world()` преобразуется в вызов `hello_world(greet)`

Этот факт объясняет особую важность параметра self и то, почему он всегда должен быть первым в любом методе объекта, который вы пишете. Вызывая метод, вы не должны передавать значение для self явно — интерпретатор сделает это за вас.

Пример

```python
class Greeter:
    def hello_world(self):
        print("Привет, Мир!")

    def greeting(self, name):
        '''Поприветствовать человека с именем name.'''
        print("Привет, {}!".format(name))

    def start_talking(self, name, weather_is_good):
        '''Поприветствовать и начать разговор с вопроса о погоде.'''
        print("Привет, {}!".format(name))
        if weather_is_good:
            print("Хорошая погода, не так ли?")
        else:
            print("Отвратительная погода, не так ли?")


greet = Greeter()
greet.hello_world()      # Привет, Мир!
greet.greeting("Петя")  # Привет, Петя!

greet.start_talking("Саша", True) # Привет, Саша!
# Хорошая погода, не так ли?
```

Инициализация экземпляров класса

```
class Car:
    def start_engine(self):
        engine_on = True  # К сожалению, не сработает

    def drive_to(self, city):
        if engine_on:  # Ошибка NameError
            print("Едем в город {}.".format(city))
        else:
            print("Машина не заведена, никуда не едем")


c = Car()
c.start_engine()
c.drive_to('Владивосток')
```

Инициализация экземпляров класса

```
class Car:
    def start_engine(self):
        self.engine_on = True  # К сожалению, не сработает

    def drive_to(self, city):
        if self.engine_on:
            print("Едем в город {}.".format(city))
        else:
            print("Машина не заведена, никуда не едем")


c = Car()
c.start_engine()
c.drive_to('Владивосток') # Едем в город Владивосток.
```

Атрибут не создан, если не вызван метод start_engine

Специальные методы

Нет ли способа задать значение атрибута engine_on по умолчанию?

Да. Есть метод __init__, который относится к группе так называемых **специальных методов**, которые имеют особое значение для интерпретатора Python.

Особое значение метода __init__ заключается в том, что если такой метод в классе определён, то интерпретатор **автоматически** вызывает его при создании каждого экземпляра этого класса для инициализации экземпляра.

Пример

```python
class Car:
    def __init__(self):
        self.engine_on = False

    def start_engine(self):
        self.engine_on = True

    def drive_to(self, city):
        if self.engine_on:
            print("Едем в город {}.".format(city))
        else:
            print("Машина не заведена, никуда не едем.")


car1 = Car()
car1.start_engine()
car1.drive_to('Владивосток')  # Едем в город Владивосток.
car2 = Car()
car2.drive_to('Лиссабон')     # Машина не заведена, никуда не едем.
```

Соглашения об именовании, вызов методов атрибутов

СтильИмёнКлассов

стиль_имён_методов_и_атрибутов

```
class RoboticMailDelivery:
    def __init__(self):
        self.house_flat_pairs = []

    def add_mail(self, house_number, flat_number):
        '''Добавить информацию о доставке письма по номеру дома
        house_number, квартира flat_number.'''
        self.house_flat_pairs.append((house_number, flat_number))

    def flat_numbers_for_house(self, house_number):
        '''Вернуть список квартир в доме house_number,
        в которые нужно доставить письма.'''
        flat_numbers = []
        for h, f in self.house_flat_pairs:
            if h == house_number:
                flat_numbers.append(f)
        return flat_numbers
```

вызов метода объекта-атрибута

Источники

1. ООП в картинках https://habr.com/ru/post/463125/
2. Объектно-ориентированное программирование

https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5

3. Объектно-ориентированное программирование (часть 1)

https://skillbox.ru/media/code/oop_chast_1_chto_takoe_klassy_i_obekty/

4. Моделирование и формализация http://www.plam.ru/compinet/osnovy_informatiki_uchebnik_dlja_vuzov/p10.php