ООП: основные принципы | Я.Шпора

ООП (объектно-ориентированное программирование) — подход в программировании, который основывается на организации кода вокруг классов и объектов с применением принципов ООП.

Принципы ООП — абстракция, наследование, полиморфизм, инкапсуляция.

Абстракция

Идея, опираясь на которую можно описывать объекты со своими характеристиками и функциями. Абстракция подразумевает использование только тех характеристик объекта, которые с достаточной точностью представляют его в задуманной системе.

Наследование

Механизм, который позволяет на основе существующих классов (родительских), создавать новые классы (дочерние). Атрибуты и методы родительских классов переиспользуются в дочерних.

Чтобы создать дочерний класс, нужно унаследовать его от родительского класса.

Синтаксис наследования:

```
class ИмяКлассаНаследника(ИмяРодительскогоКласса):
    pass
```

Пример:

```
class Parent:
    pass


class Child(Parent):
    pass
```

При наследовании можно изменять и заменять методы и атрибуты

родительского класса:

```
class Parent:
    def print_hello(self):
        print('Привет от родителя')


class Child(Parent):
    def print_hello(self):
        print('Привет от наследника')


parent = Parent()
child = Child()

parent.print_hello()
child.print_hello()

# Вывод в терминал:
# Привет от родителя
# Привет от наследника
```

> В Python все классы напрямую или через классы-родители — наследники встроенного базового класса `object`. Это значит, что все классы в Python могут использовать методы класса `object`. Например, метод `__str__()` определён именно в классе `object`.

Полиморфизм

Атрибуты и методы иногда называют интерфейсом класса. Переопределение методов позволяет сохранить интерфейс класса, изменив поведение его методов и значения атрибутов. Такое переопределение методов и атрибутов является проявлением полиморфизма в Python.

Инкапсуляция

Механизм, когда детали реализации объекта скрыты, а для взаимодействия с ним предоставлен специальный интерфейс.

Чтобы переиспользовать или переопределить метод, не нужно знать, как он реализован в родительском классе, так как метод инкапсулирован.

Защищённые атрибуты и методы

Если нужно показать, что атрибут или метод должен использоваться только для внутренних нужд класса или его наследников, перед названием атрибута или метода добавляют одинарное подчёркивание `_`. Такой атрибут или метод будет считаться защищённым.

```
class Parent:
    # Защищённый метод.
    def _print_hello(self):
        print('Привет от родителя')


parent = Parent()
# Вызвать защищённый метод.
parent._print_hello()

# Вывод в терминал:
# Привет от родителя
```

Увидев такой метод в коде, разработчик будет понимать, что желательно использовать его только внутри класса или его наследников.

Приватные атрибуты и методы

Для Python атрибут или метод, имя которого начинается с двойного подчёркивания `__`, — приватный. К этому атрибуту можно обратиться внутри класса, а вот вне класса просто так сделать это не получится.

```
class Parent:
    # Приватный метод.
    def __print_hello(self):
        print('Привет от родителя')


parent = Parent()
# Вызвать приватный метод вне класса.
```

```
parent.__print_hello()

# В терминал выведется ошибка AttributeError.
```

В Python приватные атрибуты и методы наследуются дочерними классами. Однако из-за механизма именования, известного как *name mangling* («искажение имён»), доступ к ним становится менее прямолинейным.

Когда вы определяете атрибут или метод, имя которого начинается с двойного подчёркивания, Python автоматически изменяет его имя, добавляя к нему `_ИмяКласса`. Например, если у вас есть класс `A` с приватным атрибутом `__priv_attr`, Python преобразует его имя в `_A__priv_attr`. Это делается для того, чтобы минимизировать конфликты имён при наследовании.

> В Python все уровни доступа к атрибутам и методам — это соглашение, а не строгое ограничение. Соглашение можно не соблюдать, однако это считается плохой практикой, так как нарушает принципы инкапсуляции.

В крошечной шпаргалке — великая мысль, и память умещается на ладони.

Протон Быстрый, древнегреческий спортсмен