Основы Python

GeekBrains

Урок 7

ООП. Продвинутый уровень

На этом уроке

1. Перегрузка операторов:

   __init__()

   __del__()

   __str__()

   __add__()

   __setattr__()

   __getitem__()

   __call__()

   и др.

2. Переопределение методов.
3. Интерфейсы.
4. Интерфейс итерации.
5. Собственные объекты-итераторы.
6. Декоратор @property.
7. Композиция.
8. Особенности ООП в Python.

Перегрузка операторов. Часть 1

__init__() — соответствует конструктору объектов класса, срабатывает при создании объектов.

__del__() — соответствует деструктору объектов класса, срабатывает при удалении объектов.

__str__() — срабатывает при передаче объекта функциям str() и print(), преобразует объект к строке.

__add__() — срабатывает при участии объекта в операции сложения в качестве операнда с левой стороны, обеспечивает перегрузку оператора сложения.

__setattr__() — срабатывает при выполнении операции, присваивания значения атрибуту объекта.

__getitem__() — срабатывает при извлечении элемента по индексу.

__call__() — срабатывает при обращении к экземпляру класса как к функции.

Перегрузка операторов. Часть 2

__gt__() — соответствует оператору «>».

__lt__() — соответствует оператору «<».

__ge__() — соответствует оператору «≥».

__le__() — соответствует оператору «≤».

__eq__() — соответствует оператору «==».

__iadd__() — соответствует операции «Сложение и присваивание» +=.

__isub__() — соответствует операции «Вычитание и присваивание» -=.

Переопределение методов

Специальный механизм, позволяющий использовать метод класса-родителя

в классе-потомке с добавлением некоторой функциональности.

```
class ParentClass:
    def __init__(self):
        print("Конструктор класса-родителя")

    def my_method(self):
        print("Метод my_method() класса ParentClass")


class ChildClass(ParentClass):
    def __init__(self):
        print("Конструктор дочернего класса")
        ParentClass.__init__(self)

    def my_method(self):
        print("Метод my_method() класса ChildClass")
        ParentClass.my_method(self)
```

Интерфейсы

Под интерфейсом в ООП понимается описание поведения объекта,

то есть совокупность публичных методов объекта, которые могут применяться в других частях программы для взаимодействия с ним.

Интерфейс итерации

Под итераторами понимаются специальные объекты, обеспечивающие пошаговый доступ к данным из контейнера. В привязке к итераторам работают циклы перебора (for in), встроенные функции (map(), filter(), zip()), операция распаковки.

```
my_list = [30, 105.6, "text", True]
for el in my_list:
    print(el)
```

Работа с итераторами

Итератор в Python — объект, реализующий метод __next__ без аргументов, возвращающий очередной элемент или исключение StopIteration.

1. Вызов метода __iter__() для итерируемого объекта. Метод __iter__() возвращает объект с методом __next__().
2. Цикл for in во время каждой итерации запускает метод __next__(), который при каждом вызове возвращает очередной элемент итератора.
3. Когда элементы итераторы исчерпаны, метод __next__() завершает свою работу и генерирует исключение StopIteration.

Декоратор @property

Под декоратором в Python подразумевается функция (или класс), расширяющая логику работы другой функции. Встроенный декоратор @property позволяет работать с методом некоторого класса как с атрибутом.

Композиция

В концепции ООП существует возможность реализации композиционного подхода, в соответствии с которым создаётся класс-контейнер, включающий вызовы других классов.

Особенности ООП в Python

1) Всё в Python — это объекты. Строка, число, список, словарь, функция, класс, модуль, пакет — объекты. Даже класс — тоже объект, порождающий другие объекты (экземпляры).
2) В Python все типы данных — классы.
3) Инкапсуляция в Python формальная. В других языках программирования инкапсуляция гарантирует защиту свойства класса от прямого доступа. В Python такой доступ сохраняется.

Итоги

1. Научились перегружать встроенные методы классов для изменения их стандартного поведения.
2. Узнали, для чего нужны итераторы и как создавать собственные.
3. Познакомились с такими важными конструкциями, как декораторы.
4. Научились реализовывать в своих проектах композицию.
5. Подвели итоги нашего знакомства с ООП в Python, определив основные его особенности.