ТЕХНОАТОМ

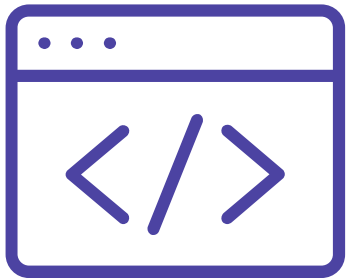
Веб-программирование Python

Лекция 2.5. Основы Python

Михалев Олег

Сегодня

- Классы
- Итераторы
- Вызываемые объекты
- Менеджеры контекста
- Дескрипторы



Объект - нечто обладающее свойствами (атрибутами) и поведением (методами). Классы - описание объектов.

В Python даже классы - объект, порождения классов правильнее называть экземплярами (instance).

Создать класс – просто!

```
1. >>> class Dummy(object):
2. ...     pass
3. ...
```

Создать экземпляр класса – еще проще!

```
1. >>> instance = Dummy()
2. >>> instance
3. <__main__.Dummy object at 0x002DEF10>
```

Классы

ТЕХНОАТОМ

```
>>> class Dummy(object):
...     def __new__(cls, *args, **kwargs):
...             instance = super().__new__(cls)
...             print('allocate memory')
...             return instance
...     def __init__(self, value):
...             self.value = value
...             print('init instance')
...     def __del__(self):
...             print('deallocate memory')
...
```

```
1. >>> instance = Dummy(10)
2. allocate memory
3. init instance
4. >>> instance = 0
5. deallocate memory
```

Метод __**new**__ создает экземпляр класса, на вход он получает **cls** - собственный класс и аргументы, которые впоследствии будут переданы в __**init**__.

Метод __**init**__ инициализирует экземпляр, на вход он получает **self** - объект и те аргументы, с которыми был вызван конструктор.

Метод **__del__** запускается при удалении экземпляра из памяти (сборщиком мусора), по сути **__del__** - деструктор объекта.

Атрибуты класса и атрибуты экземпляра - не одно и то же!

```
1. >>> class Example:
2. ...     class_var = 6
3. ...
4. >>> example = Example()
5. >>> example.class_var
6. 6
7. >>> Example.class_var = 4
8. >>> example.class_var
9. 4
10.>>> example.class_var = 2
11.>>> Example.class_var
12.4
```

Методы - это функции, принимающие ссылку на экземпляр и аргументы (как и любые функции).

```
1. >>> class Example:
2. ...     def method(self, *args, **kwargs):
3. ...             print('called')
4. ...
5. >>> example = Example()
6. >>> example.method()
7. called
```

Методы класса - это функции, принимающие ссылку на класс и аргументы (как и любые функции). Даже при вызове из экземпляра доступа к объекту экземпляра не будет.

```
1. >>> class Example:
2. ...     @classmethod
3. ...     def method(cls, *args, **kwargs):
4. ...             print('called')
5. ...
6. >>> Example.method()
7. called
```

Статические методы не получают ссылок и аналогичны обычным функциям.

```
>>> class Example:
...     @staticmethod
...     def method(*args, **kwargs):
...             print('called')
...
```

Вычисляемые атрибуты задаются с помощью **property**. Наиболее часто применяются вычисляемые атрибуты "только для чтения.

```
>>> class PropertyExample:
...     def __init__(self, value):
...             self._value = value
...     value = property()
...     @value.getter
...     def value(self):
...             return self._value
...     @value.setter
...     def value(self, value):
...             self._value = value
...     @value.deleter
...     def value(self):
...             self._value = None
...
```

```
>>> class PropertyExample:
...     def __init__(self, value):
...             self._value = value
...     @property
...     def value(self):
...             return self._value
...
>>> example = PropertyExample(5)
>>> example.value
5
>>> example.value = 0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: can't set attribute
```

Инкапсуляция
Наследование
Полиморфизм

Инкапсуляция в Python - соглашение между программистами.

```
1. >>> class Dummy:
2. ...     def _private(self):
3. ...             pass
4. ...     def __private(self):
5. ...             pass
6. ...
```

Наследование в Python выгодно отличается от других языков, множественное наследование разрешается.

Для решения конфликтов (ромбовидное наследование) используется линеаризация, родительские классы идут в порядке приоритета и усиления своей специфики.

Классы

ТЕХНОАТОМ

```
1. >>> class Cat:
2. ...     pass
3. ...
4. >>> class Dog:
5. ...     pass
6. ...
7. >>> class CatDog(Cat, Dog):
8. ...     pass
9. ...
```

Функция super обращается к атрибуту __mro__ класса. В случае перегрузки методов, функция super не вызывается.

```
>>> CatDog.__mro__
(
  <class '__main__.CatDog'>,
  <class '__main__.Cat'>,
  <class '__main__.Dog',
  <class 'object'>
)
```

Перегрузка операторов сравнения

__cmp__	полное сравнение
__eq__	равно
__ne__	не равно
__lt__	меньше
__gt__	больше
__le__	меньше или равно
__ge__	больше или равно

Перегрузка унарных арифметических операций

__pos__	унарный плюс
__neg__	унарный минус
__abs__	абсолютное значение

Перегрузка арифметических операций

__add__	сложение
__sub__	вычитание
__mul__	умножение
__floordiv__	целочисленное деление
__div__	деление
__mod__	остаток от деления
__pow__	возведение в степень

Перегрузка битовых операций

__invert__	бинарный "Не" (унарный)
__lshift__	бинарный сдвиг влево
__rshift__	бинарный сдвиг вправо
__and__	бинарный "И"
__or__	бинарный "Или"
__xor__	бинарный "Исключающее Или"

Перегрузка представлений

__str__	строковое представление
__repr__	строковое представление (не для людей)
__hash__	целочисленный хэш (не для людей)

Важно отметить, что менять **__hash__** без переопределения операции сравнения не стоит.

a == b подразумевает **hash(a) == hash(b)**.

Контроль доступа к атрибутам

__getattr__(self, name)	чтение атрибута
__setattr__(self, name, value)	запись атрибута
__delattr__(self, name)	удаление атрибута

```
>>> class AttributeAccess:
...     def __getattr__(self, name):
...             print('getattr %s' % name)
...     def __getattribute__(self, name):
...             print('getattribute %s' % name)
...             return super().__getattribute__(name)
...
>>> a = AttributeAccess()
>>> a.__class__
getattribute __class__
<class '__main__.AttributeAccess'>
>>> a.unknown
getattribute unknown
getattr unknown
None
```

Итераторы

ТЕХНОАТОМ

```
>>> class Iterable:
...     def __init__(self, n):
...         self.current = 0
...         self.limit = n
...     def __iter__(self):
...         return self
...     def __next__(self):
...         if self.current > self.limit:
...             self.current = 0
...             raise StopIteration
...         next = self.current
...         self.current += 1
...         return next
...
```

Итераторы

ТЕХНОАТОМ

```
1. >>> example = Iterable(5)
2. >>> for i in example:
3. ...     print(i)
4. ...
5. 0
6. 1
7. 2
8. 3
9. 4
10.5
```

Вызываемые объекты

ТЕХНОАТОМ

```
1. >>> class Functor:
2. ...     def __init__(self, multiplier):
3. ...         self.multiplier = multiplier
4. ...     def __call__(self, number):
5. ...         return self.multiplier * number
6. ...
7. >>> example = Functor(4)
8. >>> example(5)
9. 20
```

Менеджеры контекста

ТЕХНОАТОМ

В Python можно использовать менеджеры контекста, для этого определено ключевое слово **with**.

```
1. >>> with open('file.txt') as input_file:
2. ...     pass
3. ...
```

Менеджеры контекста

ТЕХНОАТОМ

```
>>> class FileReader:
...     def __init__(self, path):
...         self.file = open(path)
...     def __enter__(self):
...         return self.file
...     def __exit__(self, exception_type, exception, trace)
...         self.file.close()
...         return True
...
```

В Python можно создавать дескрипторы, они позволяют добавить свою логику к событиям доступа (получение, изменение, удаление) к атрибутам других объектов.

Менеджеры контекста

ТЕХНОАТОМ

```
1. >>> class Money:
2. ...     def __init__(self):
3. ...             self.value = 0
4. ...     def __get__(self, instance, instance_class)
5. ...             return round(self.value, 2)
6. ...     def __set__(self, instance, value):
7. ...             self.value = round(value, 2)
8. ...
9. >>> class PiggyBank:
10....     value = Money()
11....
12.>>> example = PiggyBank()
13.>>> example.value = 235.324234
14.>>> example.value
15.235.32
```

ТЕХНОАТОМ

Спасибо за внимание!

Михалев Олег
mailto:mhalairt@gmail.com

Домашнее задание

ТЕХНОАТОМ

Разработать класс **Charge** (денежная транзакция):

- содержащий вычисляемый атрибут **value** - вещественное число, определяющее сумму денежной транзакции;
- содержащий метод инициализации.

После инициализации экземпляра класса, значение **value** изменять нельзя. Отрицательные значения соответствуют расходам, положительные - зачислениям на счет.

Домашнее задание

ТЕХНОАТОМ

Несмотря на реальные значения суммы транзакции, вычисляемые атрибут **value** не должен отдавать значения больше чем с двумя знаками после запятой. Используйте функцию **round** и либо декоратор, либо дескриптор на свое усмотрение.

Домашнее задание

ТЕХНОАТОМ

Разработать класс **Account** (банковский счет):

- содержащий атрибут **charges** - список, определяющий последовательность денежных транзакций по счету;
- содержащий атрибут **total** - вещественное число, определяющее текущее состояние счета;
- содержащий метод инициализации;
- содержащий методы добавления транзакций (приход и расход соответственно).

Каждый элемент списка **charges** - экземпляр класса Charge.

Домашнее задание

При инициализации экземпляра класса возможно задать начальное значение для состояния счета, по умолчанию - ноль. Атрибут **total** должен обновляться при вызове методов зачисления и списания, отрицательное значение атрибута невозможны.

Дополнительно:
Класс **Account** должен быть итерабелен (отдавать **charges**).