

Углубленный Python

Лекция 3

Объектная модель, введение в ООП

Кандауров Геннадий



education

Напоминание отметиться на портале

+ ОСТАВИТЬ ОТЗЫВ

vk образование

БлогиЛюдиПрограммаВакансииРасписание

Q<

VK

Техно

Открыт приём заявок!

чт, 8 сентября

Нет занятий

пт, 9 сентября

18:00 Углубленный Py... с3
Введение в Python, основные
понятия, тестирование
Г. Кандауров

сб, 10 сентября

Нет занятий

вс, 11 сентября

Нет занятий

пн, 12 сентября

Нет занятий

Углубленный Python

↓ 0 ↑

Блог для материалов по курсу "Углубленный Python"

57 читателей, 2 топика

ПодписатьсяСоздать топик

Поиск по авторам, заголовку и тексту топика...

Найти

Добро пожаловать на курс!

Углубленный Python

ИзменитьУдалить

Всем привет и добро пожаловать на курс по углубленному изучению Python!

Прямой эфир

МоиВсе

Геннадий Кандауров час назад
Углубленный Python → Добро пожаловать
на курс! 0

Екатерина Черкасова 7 дней назад
Стажировка → Приглашаем мобильных,
фронтенд- и бэкэнд-разработчиков на
Weekend Offer! 0

Дарья Вовченко 9 дней назад
Углубленный Python → Добро пожаловать
в образовательные проекты VK
Образование! 0

Дарья Вовченко 9 дней назад
Разработка веб-сервисов на

Квиз про прошлой лекции



Содержание занятия

1. Пространства имен
2. Пример тестирования
3. Классы
4. Объектная модель
5. ООП

Пространства имен

*“Namespaces are one honking great
idea -- let's do more of those!”*

Tim Peters (import this)



Пространства имен

Пространство имен—это совокупность определенных в настоящий момент символических имен и информации об объектах, на которые они ссылаются.

- Встроенное
- Глобальное
- Объемлющее
- Локальное

Область видимости: LEGB

Область видимости имени это часть программы, в которой данное имя обладает значением.

Интерпретатор определяет эту область в среде выполнения, основываясь на том, где располагается определение имени и из какого места в коде на него ссылаются.

- | | |
|---------------|--------------------------|
| 1. Локальная | ○ <code>globals()</code> |
| 2. Объемлющая | ○ <code>locals()</code> |
| 3. Глобальная | ○ <code>global</code> |
| 4. Встроенная | ○ <code>nonlocal</code> |

__builtins__

```
>>> hasattr(__builtins__, "dir")
```

```
True
```

```
>>> dir(__builtins__)
```

```
...
```

int	map	reversed	dir
float	zip	len	type
str	filter	sum	isinstance
bool	range	all	issubclass
tuple	enumerate	any	hasattr
list	sorted	globals	getattr
dict	min	locals	setattr
set	max	callable	delattr

Классы



Классы: атрибуты

```
class A:
    name = "cls_name"
    __cls_private = "cls_private"

    def __init__(self, val):
        self.val = val
        self._protected = "protected"
        self.__private = "private"

    def print(self):
        print(
            f"{self.val=}, {self._protected=}, {self.__private=}, "
            f"{self.name=}, {self.__cls_private=}"
        )
```

Классы: методы

```
class A:
    @staticmethod
    def print_static():
        print("static")

    @classmethod
    def print_cls(cls):
        print(f"class_method for {cls.__name__}")

    def __init__(self, val):
        self.val = val

    def print_offset(self, offset=10):
        print(self.val + offset)

    def __str__(self):
        return f"{self.__class__.__name__}:val={self.val}"
```

Классы

```
object.__new__(cls[, ...])
```

Статический метод, создает новый экземпляр класса.

После создание экземпляра вызывается (уже у экземпляра) метод `__init__`.

`__init__` ничего не должен возвращать (кроме `None`), иначе - `TypeError`

```
class Singleton:
```

```
    _instance = None
```

```
    def __new__(cls, *args, **kwargs):
```

```
        if cls._instance is None:
```

```
            cls._instance = super().__new__(cls, *args, **kwargs)
```

```
        return cls._instance
```

Классы

`object.__del__(self)`

Финализатор

```
class Connector:
    def __init__(self, db_name):
        self.conn = DbDriver(db_name)
    def __del__(self):
        self.conn.close()
        print("DEL")
```

```
db = Connector("users")
```

```
del db # ???
```

Классы: свойства

классический подход

```
class Author:
```

```
    def __init__(self, name):  
        self.__name = ""  
        self.set_name(name)
```

```
    def get_name(self):  
        return self.__name
```

```
    def set_name(self, val):  
        self.__name = val
```

pythonic

```
class Author:
```

```
    def __init__(self, name):  
        self.name = name
```

```
    @property  
    def name(self):  
        return self.__name
```

```
    @name.setter  
    def name(self, val):  
        self.__name = val
```

Классы: свойства

```
class Author:
    def __init__(self, name):
        self.name = name

    @property
    def name(self):
        """name doc"""
        return self.__name

    @name.setter
    def name(self, val):
        self.__name = val

    @name.deleter
    def name(self):
        del self.__name
```

```
class Author:
    def __init__(self, name):
        self.name = name

    def get_name(self):
        return self.__name

    def set_name(self, val):
        self.__name = val

    def del_name(self):
        del self.__name

    name = property(
        get_name,
        set_name,
        del_name,
        "name doc",
    )
```

Классы: свойства read/write only

```
class Author:
    def __init__(self, name, password):
        self.__name = name
        self.password_hash = None
        self.password = password

    @property
    def name(self):
        """name is read-only"""
        return self.__name

    @property
    def password(self):
        raise AttributeError("Password is write-only")

    @password.setter
    def password(self, plaintext):
        self.password_hash = make_hash_from_password(plaintext)
```


Классы: доступ к атрибутам

Чтобы найти атрибут объекта `obj`, python обыскивает:

1. Сам объект (`obj.__dict__` и его системные атрибуты)
2. Класс объекта (`obj.__class__.__dict__`).
3. Классы, от которых унаследован класс объекта (`obj.__class__.__mro__`)

Классы: наследование

```
class Timing:
    def __init__(self, start, end):
        self.start = start
        self.end = end

    def duration(self):
        print("Timing.duration")
        return self.end - self.start
```

```
class MinuteTiming(Timing):
    def duration(self):
        print("MinuteTiming.duration")
        seconds = super().duration()
        return seconds / 60
```

```
>>> m = MinuteTiming(1000, 7000)
```

```
>>> m.duration()
```

```
MinuteTiming.duration
```

```
Timing.duration
```

```
100.0
```

Классы: super

super()

super(type, object_or_type=None)

Возвращает прокси-объект, который делегирует вызовы методов родительскому или родственному классу type.

Полезно для доступа к унаследованным методам, которые были переопределены в классе.

- mro - линейный порядок поиска атрибутов
- C3-линеаризация - алгоритм вычисления mro

Классы: MRO

Порядок разрешения методов (method resolution order) позволяет python выяснить, из какого класса-предка нужно вызывать метод, если он не обнаружен непосредственно в классе-потомке.

```
cls.__mro__
```

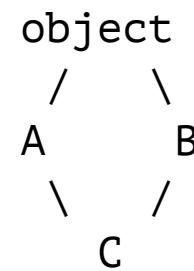
```
cls.mro()
```

```
>>> MinuteTiming.mro()
```

```
[__main__.MinuteTiming, __main__.Timing, object]
```

Классы: локальный порядок старшинства

```
>>> class A:
...     pass
...
>>> class B:
...     pass
...
>>> class C(A, B):
...     pass
...
>>> C.mro()
[<class '__main__.C'>, <class '__main__.A'>, <class '__main__.B'>, <class 'object'>]
>>>
>>> class C(B, A):
...     pass
...
>>> C.mro()
[<class '__main__.C'>, <class '__main__.B'>, <class '__main__.A'>, <class 'object'>]
```



```
graph TD
    object --> A
    object --> B
    A --> C
    B --> C
```

Классы: множественное наследование



Классы: магические атрибуты

Классы

`__name__` — имя класса

`__module__` — модуль, в котором объявлен класс

`__qualname__` — fully qualified имя

`__doc__` — докстринг

`__annotations__` — аннотации статических полей класса

`__dict__` — namespace класса

Методы

`__self__` — объект класса

`__func__` — сама функция, которую мы в классе объявили

Классы: магические атрибуты

Поля, относящиеся к наследованию

`__bases__` — базовые классы

`__base__` — базовый класс, который указан первым по порядку

`__mro__` — список классов, упорядоченный по вызову функции `super`

```
class B(A): pass
```

```
>>> B.__bases__
```

```
(__main__.A,)
```

```
>>> B.__base__
```

```
__main__.A
```

```
>>> B.__mro__
```

```
(__main__.B, __main__.A, object)
```


Классы: магические методы

Доступ к атрибутам

- `__getattribute__(self, name)`
- `__getattr__(self, name)`
- `__setattr__(self, name, val)`
- `__delattr__(self, name)`
- `__dir__(self)`

Классы: магические методы

```
object.__call__(self[, args...])
```

```
class Adder:  
    def __init__(self, val):  
        self.val = val  
  
    def __call__(self, value):  
        return self.val + value
```

```
a = Adder(10)
```

```
a(5)  # 15
```

Классы: магические методы

To string

`__repr__` — представление объекта. Если возможно, должно быть валидное python выражение для создание такого же объекта

`__str__` — вызывается функциями `str`, `format`, `print`

`__format__` — вызывается при форматировании строки

Классы: магические методы

Сравнение

`object.__lt__(self, other)`

`object.__le__(self, other)`

`object.__eq__(self, other)`

`object.__ne__(self, other)`

`object.__gt__(self, other)`

`object.__ge__(self, other)`

`x < y == x.__lt__(y) # <=, ==, !=, >, >=`

Классы: магические методы

Эмуляция чисел

```
object.__add__(self, other)
object.__sub__(self, other)
object.__mul__(self, other)
object.__matmul__(self, other) (@)
object.__truediv__(self, other)
object.__floordiv__(self, other)
object.__mod__(self, other)
object.__divmod__(self, other)
object.__pow__(self, other[, modulo])
object.__lshift__(self, other)
object.__rshift__(self, other)
object.__and__(self, other)
object.__xor__(self, other)
object.__or__(self, other)
```

Классы: магические методы

Эмуляция чисел

Методы вызываются, когда выполняются операции (+, -, *, @, /, //, %, divmod(), pow(), **, <<, >>, &, ^, |) над объектами

`x + y == x.__add__(y)`

Есть все такие же с префиксом `r` и `i`:

`__radd__` - вызывается, если левый операнд не поддерживает `__add__`

`__iadd__` - вызывается, когда `x += y`

Классы: магические методы

Эмуляция контейнеров

`object.__len__(self)`

`object.__length_hint__(self)`

`object.__getitem__(self, key)`

`object.__setitem__(self, key, value)`

`object.__delitem__(self, key)`

`object.__missing__(self, key)`

`object.__iter__(self)`

`object.__next__(self)`

`object.__reversed__(self)`

`object.__contains__(self, item)`

Классы: магические методы

`object.__hash__(self)`

Вызывается функцией `hash()` и коллекциями, которые построены на основе hash-таблиц.

Нужно, чтобы у равных объектов был одинаковый hash.

Если определен метод `__eq__` и не определен `__hash__`, то объект не может быть ключом в hashable коллекции.

```
>>> key1 = (1, 2, 3)
```

```
>>> key2 = (1, 2, 3, [4, 5])
```

```
>>> s = set()
```

```
>>> s.add(key1) # ???
```

```
>>> s.add(key2) # ???
```


Классы: магические методы

`object.__slots__`

Позволяет явно указать поля, которые будут в классе.

В случае указания `__slots__` пропадают поля `__dict__` и `__weakref__`.

Используя `__slots__` можно экономить на памяти и времени доступа к атрибутам объекта.

```
class Point:
    __slots__ = ("x", "y")
    def __init__(self, x, y):
        self.x = x
        self.y = y
```

Классы: `__init_subclass__`

```
class Timing:
    def __init__(self, start, end):
        self.start = start
        self.end = end

    @classmethod
    def __init_subclass__(cls, **kwargs):
        print("INIT subclass", cls, kwargs)
```

```
class MinuteTiming(Timing):
    def duration(self):
        print("MinuteTiming.duration")
        seconds = super().duration()
        return seconds / 60
```

Домашнее задание #03

- Реализовать кастомный список, унаследованный от `list`
- +тесты
- `flake8` + `pylint` перед сдачей



Напоминание отметиться на портале Vol 2

+ ОСТАВИТЬ ОТЗЫВ ПОСЛЕ ЛЕКЦИИ

vk образование

БлогиЛюдиПрограммаВакансииРасписание

Q<

VK Техно

Открыт приём заявок!

чт, 8 сентября

Нет занятий

пт, 9 сентября

18:00 Углубленный Py... с3
Введение в Python, основные
понятия, тестирование
Г. Кандауров

сб, 10 сентября

Нет занятий

вс, 11 сентября

Нет занятий

пн, 12 сентября

Нет занятий

Углубленный Python

↓ 0 ↑

Блог для материалов по курсу "Углубленный Python"

57 читателей, 2 топика

ПодписатьсяСоздать топик

Поиск по авторам, заголовку и тексту топика...

Найти

Добро пожаловать на курс!

Углубленный Python

ИзменитьУдалить

Всем привет и добро пожаловать на курс по углубленному изучению Python!

Прямой эфир

МоиВсе

Геннадий Кандауров час назад
Углубленный Python → Добро пожаловать
на курс! 0

Екатерина Черкасова 7 дней назад
Стажировка → Приглашаем мобильных,
фронтенд- и бэкэнд-разработчиков на
Weekend Offer! 0

Дарья Вовченко 9 дней назад
Углубленный Python → Добро пожаловать
в образовательные проекты VK
Образование! 0

Дарья Вовченко 9 дней назад
Разработка веб-сервисов на

Спасибо за
внимание

