ТЕХНОАТОМ

Объектная модель

Лекция 2

Алена Елизарова

Про что поговорим

ТЕХНОАТОМ

1. Стандартные типы в питоне
2. Магические поля объектов
3. Магические методы объектов
4. Методы кастомизации доступа к атрибутам
5. Методы кастомизации классов

Всё есть объект

ТЕХНОАТОМ

“Objects are Python’s abstraction for data. All data in a Python program is represented by objects or by relations between objects.”

docs.python.org

Объект

ТЕХНОАТОМ

1. Каждый объект имеет id, тип и значение
2. Id никогда не меняется после создания объекта (is сравнивает id объектов)
3. Тип объекта определяет какие операции с ним можно делать
4. Значение объекта может меняться

все стандартные типы в Python
https://github.com/python/cpython/blob/ab67281e95de1a88c4379a75a547f19a8ba5ec30/Objects/object.c#L1720

ТЕХНОАТОМ

Типы с одним значением

- None
- NotImplemented
- Ellipsis (…)

больше про NotImplemented
https://docs.python.org/3/library/numbers.html#implementing-the-arithmetic-operations

Типы с одним значением

ТЕХНОАТОМ

```
>>> None
>>> type(None)
<class 'NoneType'>

>>> NotImplemented
NotImplemented
>>> type(NotImplemented)
<class 'NotImplementedType'>

>>> ...
Ellipsis
>>> type(...)
<class 'ellipsis'>

>>> type(None)()
>>> type(None)() is None
True
>>> type(NotImplemented)() is NotImplemented
True
```

Стандартные типы

ТЕХНОАТОМ

numbers.Number

- numbers.Integral (int, bool)
- numbers.Real (float)
- numbers.Complex (complex)

numbers.Number

TEXHOATOM

```
>>> import numbers
>>> issubclass(int, numbers.Number)
True

>>> issubclass(bool, int)
True

>>> issubclass(float, numbers.Real)
True
```

Sequences

Представляют собой конечные упорядоченные множества, которые проиндексированы неотрицательными числами
Делятся на:
immutable - Strings, Tuples, Bytes
mutable - Lists, Byte Arrays

Set

Множество уникальных неизменяемых объектов. По множеству не индексируется, но по нему можно итерироваться
Существует 2 типа множеств: Sets, Frozen sets

Mappings

Есть только 1 маппинг тип – Dictionaries. Ключами могут быть только неизменяемые типы, также стоит отметить, что hash от ключа должен выполняться за константное время, чтобы структура данных была эффективной.

Стандартные типы

ТЕХНОАТОМ

Подумать

```
>>> a = {1.0}

>>> 1.0 in a
???

>>> 1 in a
???

>>> True in a
???
```

Модули

Модули являются основным компонентом организации кода в питоне (и это тоже объекты).

Callable types

- Пользовательские функции
- Методы класса
- Корутины
- Асинхронные генераторы
- Built-in methods
- Классы
- Экземпляры класса

Пользовательские функции

__doc__ докстринг, изменяемое

__name__ имя функции, изменяемое

__qualname__ fully qualified имя, изменяемое

__module__ имя модуля, в котором определена функция, изменяемое

Магические поля

ТЕХНОАТОМ

Пользовательские функции

```
>>> def foo():
...     """aaaaaaa"""
...     pass
...
>>> foo.__doc__
'aaaaaaa'
>>> foo.__name__
'foo'
>>> def wrapper():
...     a = 1
...     def foo():
...             print(a)
...     return foo
...
>>> wrapper().__qualname__
'wrapper.<locals>.foo'
>>> wrapper.__module__
'__main__'
```

Пользовательские функции

__defaults__ tuple дефолтных значений, изменяемое

__code__ объект типа code, изменяемое

__globals__ словарь глобальных значений модуля, где функция объявлена, неизменяемое

__dict__ namespace функции, изменяемое

Магические поля

ТЕХНОАТОМ

Пользовательские функции

```
>>> def foo(a=1, b=2):
...     pass
...

>>> foo.__defaults__
(1, 2)

>>> foo.__code__
<code object foo at 0x7f98fe73d660, file "<stdin>", line 1>

>>> foo.__globals__
{…'__name__': '__main__', 'numbers': <module 'numbers' from
'/usr/local/lib/python3.7/numbers.py'>…}

>>> foo.a = 1
>>> foo.__dict__
{'a': 1}
```

ТЕХНОАТОМ

Пользовательские функции

__annotations__ словарь аннотаций, изменяемое

__kwdefaults__ словарь дефолтных значений кваргов, изменяемое

Магические поля

ТЕХНОАТОМ

Пользовательские функции

```
>>> def foo(a: int, b: float):
...     pass
...
>>> foo.__annotations__
{'a': <class 'int'>, 'b': <class 'float'>}

 >>> def foo(*, a=1, b=2):
...     pass
...
>>> foo.__kwdefaults__
{'a': 1, 'b': 2}
```

Пользовательские функции

__closure__ tuple ячеек, которые содержат биндинг к переменным замыкания

Магические поля

ТЕХНОАТОМ

Классы

__name__ имя класса

__module__ модуль, в котором объявлен класс

__qualname__ fully qualified имя

__doc__ докстринг

__annotations__ аннотации статических полей класса

__dict__ namespace класса

Магические поля

ТЕХНОАТОМ

__self__ объект класса

__func__ сама функция, которую мы в классе объявили

Магические поля

ТЕХНОАТОМ

Методы

```
>>> class A:
...     def foo():
...         pass
...
>>> A.foo
<function A.foo at 0x1025929d8>
>>> A().foo
<bound method A.foo of <__main__.A object at 0x102595048>>
>>> A().foo.__func__
<function A.foo at 0x1025929d8>
>>> A().foo.__self__
<__main__.A object at 0x102595048>
```

Магические поля

ТЕХНОАТОМ

Классы (поля, относящиеся к наследованию)

__bases__ базовые классы

__base__ базовый класс, который указан первым по порядку

__mro__ список классов, упорядоченный по вызову super функции

Классы (внутренности интерпретатора)

__dictoffset__

__flags__

__itemsize__

__basicsize__

__weakrefoffset__

Классы __slots__

Поле позволяет явно указать поля, которые будут в классе. В случае указания __slots__ пропадают поля __dict__ и __weakref__

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

Магические поля

ТЕХНОАТОМ

Класс может реализовывать определенные операции, которые вызываются специальным синтаксисом (например, арифметические операции).
Это подход используется в Python к перегрузке операторов.

Доступ к атрибутам

Рассмотрим подробнее атрибут __dict__
Чтобы найти атрибут объекта o, python обыскивает:
1) Сам объект (o.__dict__ и его системные атрибуты).
2) Класс объекта (o.__class__.__dict__).
3) Классы, от которых наследован класс объекта (o.__class__.__mro__.__dict__).

>>> test_dict.py

Магические поля

ТЕХНОАТОМ

```
>>> class A:
...     def foo(self):
...         pass
...
>>> a = A()
>>> A.__dict__
mappingproxy({…'foo': <function A.foo at 0x100f5af28>…})

>>> A.foo
<function A.foo at 0x100f5af28>

>>> a.foo
<bound method A.foo of <__main__.A object at 0x100f6bf98>>
```

Магические поля

ТЕХНОАТОМ

Дескрипторы

```
>>> a.foo.__class__.__get__
<slot wrapper '__get__' of 'method' objects>

>>> a.foo.__func__
<function A.foo at 0x100f5af28>
```

Магические поля

ТЕХНОАТОМ

“In general, a descriptor is an object attribute with “binding behavior”, one whose attribute access has been overridden by methods in the descriptor protocol. Those methods are __get__(), __set__(), and __delete__(). If any of those methods are defined for an object, it is said to be a descriptor.”

docs.python.org

Дескрипторы

Если определен один из методов на предыдущем слайде - объект считает дескриптором.
Если объект дескриптора определяет __**get**__, __**set**__ - он считает data дескриптором.
Если объект дескриптора определяет __**get**__ - он считает non-data дескриптор.
Они отличаются приоритетом вызова по отношению к полю __**dict**__

Методы доступа к атрибутам (yet another магия)

Методы __getattr__(), __setattr__(), __delattr__() и __getattribute__().
В отличие от дескрипторов их следует определять для объекта, содержащего атрибуты и вызываются они при доступе к любому атрибуту этого объекта.

__getattribute__(self, name)

будет вызван при попытке получить значение атрибута. Если этот метод переопределён, стандартный механизм поиска значения атрибута не будет задействован.

__setattr__(self, name, value)

будет вызван при попытке установить значение атрибута экземпляра. Аналогично __getattribute__(), если этот метод переопределён, стандартный механизм установки значения не будет задействован

__delattr__(self, name)

аналогичен __setattr__(), но используется при удалении атрибута.

Итого по методам доступа 1

ТЕХНОАТОМ

Чтобы получить значение атрибута attrname:

- Если определён метод a.__class__.__getattribute__(), то вызывается он и возвращается полученное значение.
- Если attrname это специальный (определённый python-ом) атрибут, такой как __class__ или __doc__, возвращается его значение.
- Проверяется a.__class__.__dict__ на наличие записи с attrname. Если она существует и значением является data дескриптор, возвращается результат вызова метода __get__() дескриптора. Также проверяются все базовые классы.

Итого по методам доступа 2

ТЕХНОАТОМ

- Если в a.__dict__ существует запись с именем attrname, возвращается значение этой записи.
- Проверяется a.__class__.__dict__, если в нём существует запись с attrname и это non-data дескриптор, возвращается результат __get__() дескриптора, если запись существует и там не дескриптор, возвращается значение записи. Также обыскиваются базовые классы.
- Если существует метод a.__class__.__getattr__(), он вызывается и возвращается его результат. Если такого метода нет — выкидывается AttributeError.

Итого по методам доступа 3

ТЕХНОАТОМ

Чтобы установить значение value атрибута attrname экземпляра a:

- Если существует метод a.__class__.__setattr__(), он вызывается.
- Проверяется a.__class__.__dict__, если в нём есть запись с attrname и это дескриптор данных — вызывается метод __set__() дескриптора. Также проверяются базовые классы.
- a.__dict__ добавляется запись value с ключом attrname.

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

Rich comparison

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), <=, ==, !=, >, >=

__hash__

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. __hash__ может быть определен только у неизменяемых типов

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.__reversed__(self)
object.__contains__(self, item)

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)

Эмуляция чисел

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)

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

Эмуляция чисел

object.__neg__(self)
object.__pos__(self)
object.__abs__(self)
object.__invert__(self)

Домашнее задание

1. Придумать и сделать дз по материалу первой лекции
2. Реализовать класс, отнаследованный от списка, такой, что один список
   - можно вычитать из другого
   [5, 1, 3] - [1, 2, 7] = [4, -1, -4]
   - можно складывать с другим
   - при неравной длине, дополнять меньший список нулями
   - при сравнении списков должна сравниваться сумма элементов списков

Домашнее задание

3. Написать класс для подсчета суммы в разной валюте (выбрать любые 5). Объект класса должен принимать обязательный аргумент - количество и необязательный - единица измерения (например ‘RUB’).
При сложении двух разных валют результат должен быть в валюте первого операнда.
К объекту можно прибавить число и получить результат в валюте другого операнда.
Реализовать методы __repr__ и __str__

ТЕХНОАТОМ

Спасибо за внимание!