Углубленный Python

Лекция 1

Опрышко Александр

Кто мы?

Александр Опрышко

Выпускник Технопарка

Закончил МГТУ им. Н. Э. Баумана (ИУ5). Аспирант кафедры ИУ5

В MRG работал с 2015 по 2017 год в LMS и biz.mail.ru

С 2017 году работаю в KTS Studio. Co-founder & Старший разработчик

Кто мы?

Игорь Латкин

Выпускник Технопарка

Закончил МГТУ им. Н. Э. Баумана (ИУ5). Аспирант кафедры ИУ5

В MRG работал с 2015 по 2017 год в cloud.mail.ru

С 2016 году работаю в KTS Studio. Co-founder & Старший разработчик

“

Не забудьте отметиться на занятии!

Цитата великих

Цель

Цель - подробно рассказать про особенности языка питон, которые не затрагиваются в базовых курсах, а также разобраться в инструментах и технологиях, которые используются в популярных промышленных библиотеках, например, django, asyncpg etc. А также лучших взять на стажировку в компанию.

Структура курса:

- 9 лекций/практических занятий. Половину времени мы с вами будем изучать новый материал, а другую половину - разбираться в новом ДЗ или принимать старые
- 4 ДЗ во время курса
- итоговое ДЗ

Что будет в курсе?

1. Устройство классов и объектов, разберемся как использовать, во благо, черную магию в питоне
2. Поймем как устроен интерпретатор cpython
3. Научимся работать с threads, multiprocessing
4. Разберемся как писать c-extensions и на cython
5. Асинхронное программирование
6. Затронем мелкие аспекты языка, которые всегда игнорируются (logging, работа с датами, профилирование)
7. Расскажем как публиковать свои библиотеки в pypi

Правила игры

В курсе не будет рубежных контролей, **только ДЗ**.

За выполнение и защиту каждого ДЗ ставится 15 баллов. Выполнение - 5 баллов. Защита - 10 баллов.

Срок на выполнение - 2 недели, потом за каждую неделю будут списывать 2 штрафных балла. Срок на защиту - 2 недели после выполнения.

В течение курса вы можете набрать **60 баллов**. В конце вам будет выдано **итоговое ДЗ**, которое оценивается в **40 баллов**. Выполнение - 20, защита - 20. Срок - окончание курса.

Проверка ДЗ

Отправлять нужно **своему семинаристу** с пометкой "[TP] Python {{ Фамилия Имя }}".

В письме должна быть ссылка на репозиторий (github) и список того, что было сделано/исправлено в ДЗ.

Вопросы?

Есть вопросы по организационной части?

Лекция 1. Что сегодня будет?

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

“

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

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

Стандартные типы

Типы с одним значение

- None
- NotImplemented
- Ellipsis (…)

Типы с одним значение


```
1.  >>> None
2.  >>> type(None)
3.  <class 'NoneType'>

4.  >>> NotImplemented
5.  NotImplemented
6.  >>> type(NotImplemented)
7.  <class 'NotImplementedType'>

8.  >>> ...
9.  Ellipsis
10. >>> type(...)
11. <class 'ellipsis'>

12. >>> type(None)()
13. >>> type(None)() is None
14. True
15. >>> type(NotImplemented)() is NotImplemented
16. True
```

Стандартные типы

numbers.Number

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

numbers.Number


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

4.  >>> issubclass(bool, int)
5.  True
6.
    >>> issubclass(float, numbers.Real)
7.  True
```

Стандартные типы

Sequences

Представляют собой конечные упорядоченные множества, которые проиндексированы неотрицательными числами

Делятся на:

- immutable - Strings, Tuples, Bytes
- mutable - Lists, Byte Arrays

Стандартные типы

Set

Множество уникальных immutable объектов. По множеству не индексируется, но по нему можно итерироваться

Существует 2 типа множеств: Sets, Frozen sets

Стандартные типы

Mappings

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

Стандартные типы

Подумать

```
1.  >>> a = {1.0}

2.  >>> 1.0 in a
3.  ???

4.  >>> 1 in a
5.  ???

6.  >>> True in a
7.  ???
```

Стандартные типы

Модули

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

Стандартные типы

Callable types

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

Магические поля

Пользовательские функции

__doc__ - докстринг, изменяемое

__name__ - имя функции, изменяемое

__qualname__ - fully qualified имя, изменяемое

__module__ - имя модуля, в котором определена функция, изменяемое

Магические поля

Пользовательские функции

```
1.  >>> def foo():
2.  ...     """aaaaaaa"""
3.  ...     pass
4.  ...
5.  >>> foo.__doc__
6.  'aaaaaaa'
7.  >>> foo.__name__
8.  'foo'
9.  >>> def wrapper():
10. ...     a = 1
11. ...     def foo():
12. ...             print(a)
13. ...     return foo
14. ...
15. >>> wrapper().__qualname__
16. 'wrapper.<locals>.foo'
17. >>> foo.__module__
18. '__main__'
```

Магические поля

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

Пользовательские функции

```
1.  >>> def foo(a=1, b=2):
2.  ...     pass
3.  ...

4.  >>> foo.__defaults__
5.  (1, 2)

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

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

10. >>> foo.a = 1
11. >>> foo.__dict__
12. {'a': 1}
13.
```

Магические поля

Пользовательские функции

__annotations__ - словарь аннотаций, изменяемое

__kwdefaults__ - словарь дефолтных значений кваргов, изменяемое

Магические поля

Пользовательские функции

```
1.  >>> def foo(a: int, b: float):
2.  ...     pass
3.  ...
4.  >>> foo.__annotations__
5.  {'a': <class 'int'>, 'b': <class 'float'>}
6.
     >>> def foo(*, a=1, b=2):
7.  ...     pass
8.  ...
9.  >>> foo.__kwdefaults__
10. {'a': 1, 'b': 2}
11.
```

Магические поля

Пользовательские функции

__closure__ - tuple ячеек, которые содержат биндинг к переменным замыкания

>>> test_closure.py

Магические поля

Методы класса

__self__ - объект класса

__func__ - сама функция, которую мы в классе объявили

Магические поля

Методы класса

```
1.  >>> class A:
2.  ...     def foo():
3.  ...         pass
4.  ...
5.  >>> A.foo
6.  <function A.foo at 0x1025929d8>
7.  >>> A().foo
8.  <bound method A.foo of <__main__.A object at 0x102595048>>
9.  >>> A().foo.__func__
10. <function A.foo at 0x1025929d8>
11. >>> A().foo.__self__
12. <__main__.A object at 0x102595048>
```

Магические поля

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

Классы (поля, относящиеся к наследованию)

__bases__ - базовые классы

__base__ - базовый класс, который указан первым по порядку

__mro__ - список классов, упорядоченный по вызову super функции

>>> test_bases.py

Магические поля

Классы (кишки интерпретатора)

__dictoffset__

__flags__

__itemsize__

__basicsize__

__weakrefoffset__

__text_signature__

Магические поля

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.

>>> test_slots.py

Магические методы

Класс может реализовывать определенные операции, которые вызываются специальным синтаксисом (например, арифметические операции или подписка и разрезание).

Это подход используется в python к перегрузке операторов.

Магические методы

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 насаледован класс объекта (o.__class__.__mro__.__dict__).

>>> test_dict.py

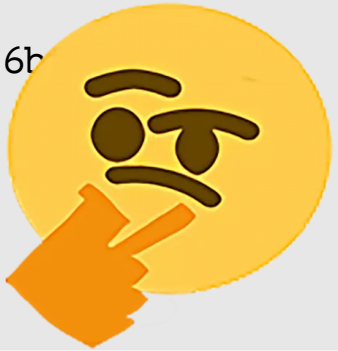
Магические методы

Подумать?

```
1.  >>> class A:
2.  ...     def foo(self):
3.  ...         pass
4.  ...
5.  >>> a = A()
6.  >>> A.__dict__
7.  mappingproxy({…'foo': <function A.foo at 0x100f5af28>…})

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

10. >>> a.foo
11. <bound method A.foo of <__main__.A object at 0x100f6b
12.
```



Магические методы

Дескрипторы

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

3.  >>> a.foo.__func__
4.  <function A.foo at 0x100f5af28>
5.
```

“

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__**

Магические методы

Примеры дескрипторов

>>> test_data_descriptor.py

>>> test_non_data_descriptor.py

>>> test_descriptor_examples.py

Некоторая оптимизация:

>>> test_binding.py

Магические методы

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.

Магические методы

__getattr__(self, name)

будет вызван в случае, если запрашиваемый атрибут не найден обычным механизмом (в __dict__ экземпляра, класса и т.д.)

Магические методы

__getattribute__(self, name)

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

Магические методы

__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

Магические методы

__delattr__(self, name)

аналогичен __setattr__(), но используется при удалении атрибута.

Примеры:

>>> test_getattr.py

Итого по методам доступа 1

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

- 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

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.

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

Магические методы

To string

```
1.  >>> class A:
2.  ...   def __str__(self):
3.  ...     return '1'
4.  ...   def __format__(self, format_spec):
5.  ...     print(format_spec)
6.  ...     return '2'
7.  ...
8.  >>> a = A()
9.  >>> print(a)
10. 1
11. >>> print(f'{a}')
12.
    2
13. >>> '%s' % a
14. '1'
15. >>> print('{a:123}'.format(a=a))
16. 123
17. 2
18.
```

Магические методы

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

>>> test_ge.py

Магические методы

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

>>> test_ge.py

Магические методы

__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 immutable типов

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.__reversed__(*self*)

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

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*)

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

Магические методы

Эмуляция чисел

object.__**neg**__(*self*)

object.__**pos**__(*self*)

object.__**abs**__(*self*)

object.__**invert**__(*self*)

Вызывается, когда выполняются унарная операция -, +, abs() and ~

>>> test_add.py

Магические методы

Кастомизация объектов

object.__**new**__(*cls*[, ...]) – создает новый объект класса, статический метод по преданию.

После создание объекта вызывается (уже у объекта) метод __**init**__. Он ничего не должен возвращать, иначе будет TypeError

>>> test_new.py

Магические методы

Кастомизация объектов

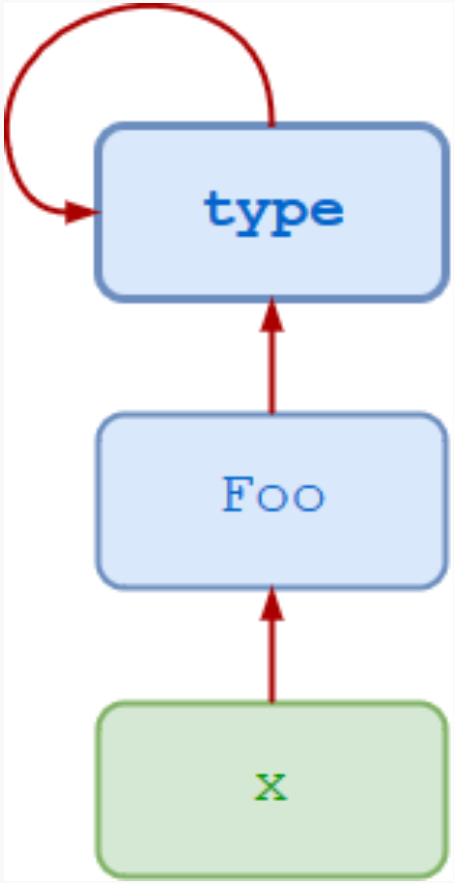
```
1.  >>> class A:
2.  ...     def __init__(self):
3.  ...         return 1
4.  ...
5.  >>> a = A()
6.  Traceback (most recent call last):
7.   File "<stdin>", line 1, in <module>
8.  TypeError: __init__() should return None, not 'int'
9.
```

Магические методы

Подумать?

```
1.  >>> class Foo:
2.  ...     pass
3.  ...
4.  >>> x = Foo()
5.  >>> type(x)
6.  <class '__main__.Foo’>
7.  >>> type(Foo)
8.  ???
9.  >>> type(type)
    ???
```

Магические методы



Магические методы

Метаклассы

Новые классы создаются с помощью вызова type(<name>, <bases>, <classdict>)

name – имя класса (__name__)

bases – базовые классы (__bases__)

classdict – namespace класса (__dict__)

Магические методы

Метаклассы

```
1.  >>> Bar = type('Bar', (Foo,), dict(attr=100))
2.  >>> x = Bar()
3.  >>> x.attr
4.  100
5.  >>> x.__class__
6.  <class '__main__.Bar’>
7.  >>> x.__class__.__bases__
8.  (<class '__main__.Foo'>,)

9.  >>> class Bar(Foo):
10. ...     attr = 100
11. ...
12. >>> x = Bar()
13. >>> x.attr
14. 100
15. >>> x.__class__.__bases__
16. (<class '__main__.Foo'>,)
```

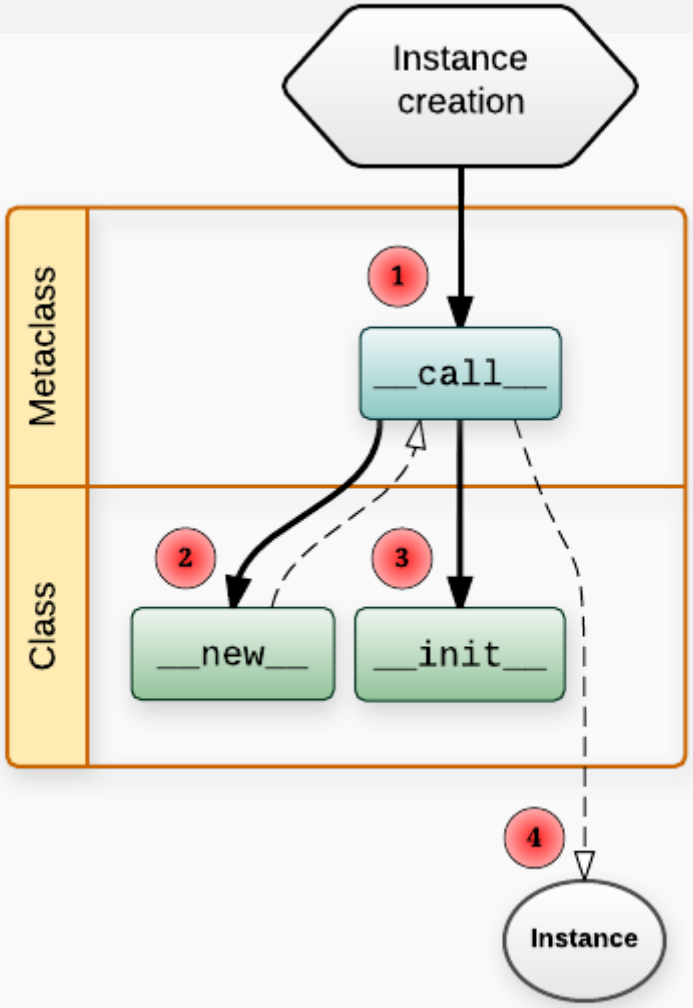
Магические методы

Кастомные метаклассы

```
1.  >>> class Meta(type):
2.  ...     Pass
3.  ...
4.  >>> class Foo(metaclass=Meta):
5.  ...     Pass
6.  ...
7.  >>> Foo()
```

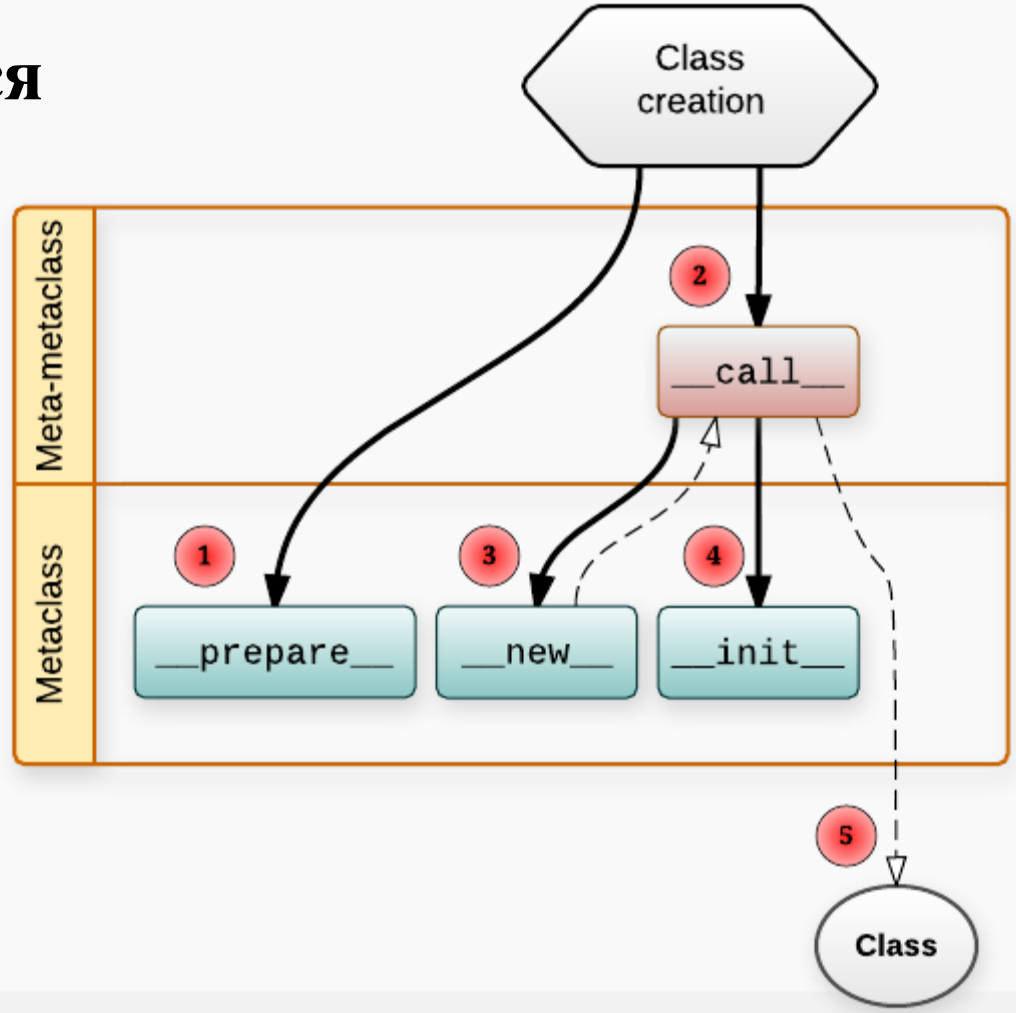
Магические методы

Повторим как происходит создание объекта



Магические методы

Как создается класс?



Магические методы

Как создается класс?

- определяются базовые классы
- определяется метакласс
- подготавливается namespace класса (__prepare__)
- выполняется тело класса
- создается класс (__new__, __init__)

>>> test_meta.py

>>> test_meta_example.py

Магические методы

Другие методы кастомизации классов

__init_subclass__

>>> test_init_subclass.py

class decorators

>>> test_ class_decorators.py

Итого

1. Стандартные типы (https://github.com/python/cpython/blob/ab67281e95de1a88c4379a75a547f19a8ba5ec30/Objects/object.c#L1720)
2. Магические поля
3. Магические методы
4. Дескрипторы
5. Метаклассы

telegram: alexopryshko
email: alexopryshko@gmail.com

telegram: igorcoding
email: igor.latkin@outlook.com

Спасибо за внимание!