17. Метаобектен протокол

27 април 2026

Питанка

Какво ще изведе следният Python код:

```
pattern = Pattern.compile("[0-9]+", Pattern.CASE_INSENSITIVE);
matcher = pattern.matcher("abc00123xyz456_0");
match = matcher.matches();
```

А така?

```
System.out.println("matches() found nothing");
```

Това е Java, обиждате ни...

Питанка \d

Какво ще изведе следният (наистина) Python код:

```
sentence = 'ще ти оставя подарък накрая'
matched = re.match(r'(\w* )+', sentence)
print(matched.group())
print(matched.groups())
```

```
'ще ти оставя подарък '
('подарък ',)
```

Питанка \d\.\d

Какво ще изведе следният (отново наистина?) Python код:

```
matches = re.finditer('\d+', '12 drummers drumming, 11 ... 10 ...')
print(matches)
```

```
<callable_iterator object at 0x00000163C89CF1F0>
```

А този?

```
for match in matches:
    print(match.span())
```

```
(0, 2)
(22, 24)
(29, 31)
```

Преговор: атрибути

```
dir(foo) -> foo.__dict__

getattr(foo, 'x') -> foo.__getattribute__('x') -> foo.__getattr__('x')

setattr(foo, 'x', 'y') -> foo.__setattr__('x', 'y')

del foo.x -> delattr(foo, 'x') -> foo.__delattr__('x')
```

Мета

- **μετά /mɛ.tə/** (гръцки)
  Представка за положение зад, през, след или отвъд нещо
- **meta /ˈmɛtə/** (английски)
  Отнасящ се до себе си или условностите на жанра си; самореферентен
- **метаобекти**
  Обекти, които описват обекти
- Други примери: метаданни, мета-информация, мета-хумор, метатип, метапрограмиране (за последното ще разкажем повече следващия път)

Протокол

(от френски: protocole; от латински: protocollum; от старогръцки: πρωτόκολλο) най-общо представлява определено множество от правила в употребление при дадени обстоятелства.

(от Wikipedia)

Метаобектен протокол

Objects are Python’s abstraction for data. All data in a Python program is represented by objects or by relations between objects. (In a sense, and in conformance to Von Neumann’s model of a “stored program computer”, code is also represented by objects.)

(из https://docs.python.org/3/reference/datamodel.html)

Метаобектен протокол (2)

- Python взаимства идеи от Lisp
- Предимство: не е нужно да знаеш Python, за да четеш Python
- Недостатък: липсва консистентността в репрезентацията на код/данни
- Прилича на прост начин за reflection
- Може да го срещнете като “MOP”

__dunders__

- Представляват (голяма част от) метаобектния протокол
- Могат да бъдат четени и/или писани и/или предефинирани
- ... и ни позволяват да интроспектираме обекти
- ... или пък да “вмъкнем” наш код на разни места в езика

__dunders__ (2)

Нека:

- Преговорим познатите
- Разгледаме някои от непознатите
- Игнорираме по-апокрифните

Кастове

- `__bool__(self)`
- `__float__(self)`
- `__int__(self)`
- `__str__(self)`

Кастове (2)

```
class Scotsman:
    def __bool__(self):
        print('Converting to bool...')
        return True

if Scotsman():
    print('A True Scotsman!')
```

```
Converting to bool...
A True Scotsman!
```

Репрезентация

- `__repr__(self)`
- `__str__(self)`
- `__doc__`
- `__dir__(self)`

Аритметика

- `__add__(self, a)`
- `__sub__(self, a)`
- `__mul__(self, a)`
- `__floordiv__(self, a)`
- `__truediv__(self, a)`
- `__divmod__(self, a)`
- `__pow__(self, a)`
- `__matmul__(self, a)`

Полиморфизъм

- Код, който се държи различно спрямо (типа на) параметрите си
- Примери от други езици:
  - generics;
  - overloading;
  - multiple dispatch (multimethods);
  - виртуални методи
- Методите в Python синтактично имат само последното
  - ... но не го наричаме така;
  - полиморфизъм само по първия аргумент (self);
  - a.к.a. single dispatch
- Операторите имплементират нещо като double dispatch чрез “десни варианти”

Аритметика (2)

“десни” варианти:

- `__radd__(self, a)`
- `__rsub__(self, a)`
- `__rmul__(self, a)`
- `__rdiv__(self, a)`
- `__rfloordiv__(self, a)`
- `__rtruediv__(self, a)`
- `__rdivmod__(self, a)`
- `__rpow__(self, a)`
- `__rmatmul__(self, a)`
- `NotImplemented`

Аритметика (3)

```
>>> class X:
...     pass
...
>>> x = X()
>>> 1 + x
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'X'
>>> (1).__add__(x)
NotImplemented
```

Аритметика (4)

```
>>> class X:
...     def __radd__(self, other):
...         print(f'called with self={self}, other={other}')
...         return self
...
>>> x = X()
>>> 1 + x
called with self=<__main__.X object at 0x7fe4be931070>, other=1
<__main__.X object at 0x7fe4be931070>
```

Аритметика (5)

“in-place” варианти:

- `__iadd__(self, a)`
- `__isub__(self, a)`
- `__imul__(self, a)`
- `__idiv__(self, a)`
- `__ifloordiv__(self, a)`
- `__itruediv__(self, a)`
- `__idivmod__(self, a)`
- `__ipow__(self, a)`
- `__imatmul__(self, a)`

Унарна аритметика

- `__abs__(self)`
- `__pos__(self)`
- `__neg__(self)`

Битова аритметика

- `__and__ / __rand__ / __iand__(self, a)`
- `__or__ / __ror__ / __ior__(self, a)`
- `__xor__ / __rxor__ / __ixor__(self, a)`
- `__rshift__ / __rrshift__ / __irshift__(self, n)`
- `__lshift__ / __rlshift__ / __ilshift__(self, n)`
- `__invert__(self)`

Равенство и хеширане

- `__eq__(self, a)`
- `__ne__(self, a)`
- `__hash__(self)`

Равенство и хеширане (2)

Имат вградено поведение:

- `x == y # x is y`
- `x != y # not (x == y)`
- `hash(x) # някаква_функция(id(x))`
  - към момента на 64 битов OS: `id(x) // 16`
  - не разчитайте поведението на последното да се запази!

Сравнения

- `__lt__(self, a)`
- `__le__(self, a)`
- `__gt__(self, a)`
- `__ge__(self, a)`

Сравнения (2)

```
>>> class Python:
...     def __gt__(self, other):
...         print('Greatest!')
...         return True
...
>>> Python() > 'Java'
Greatest!
True
```

Сравнения (3)

- Double dispatch!
- нямаме десен вариант на `__lt__`
- вместо него се ползва `__gt__`
- ... и обратното
- аналогично за `__ge__`/`__le__`

Сравнения (4)

```
>>> class Python:
...     def __gt__(self, other):
...         print('Greatest!')
...         return True
...
>>> 'Java' < Python()
Greatest!
True
>>> 'Java'.__lt__(Python())
NotImplemented
```

with

- `__enter__(self)`
- `__exit__(self, type, value, traceback)`

Атрибути

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

Колекции

- `__len__(self)`
- `__contains__(self, item)`
- `__getitem__(self, i)`
- `__setitem__(self, i, value)`
- `__delitem__(self, i)`

Итератори

- `__iter__(self)`
- `__next__(self)`

Метаатрибути

- `__dict__`
- `__slots__`
- `__class__`
- `__name__`

Функции

- `__code__`
- `__call__(self, *args, **kwargs)`

Импортиране на модули

```
import module
```

...съответства на...

```
module = __import__('module')
```

Конструиране

- `__new__(cls, *args, **kwargs)`
- `__init__(self, *args, **kwargs)`

__new__

`__new__` е истинският конструктор на вашите обекти. `__init__` е само инициализатор.

```
class Vector(tuple):
  def __new__(klass, x, y):
    return tuple.__new__(klass, (x, y))

  def __add__(self, other):
    if not isinstance(other, Vector):
      return NotImplemented
    return Vector(self[0] + other[0], self[1] + other[1])
```

Декораторен клас от предизвикателство #2

```
from collections import defaultdict

class Memnick:

    MEMORY = defaultdict(list)

    def __new__(self, *args):
        if args:
            return sum((self.MEMORY.get(arg.__name__, []) for arg in args), [])
        return super().__new__(self)

    def __call__(self, func):
        self._func = func
        self._decorated.__func__.__name__ = func.__name__
        return self._decorated

    def _decorated(self, *args, **kwargs):
        result = self._func(*args, **kwargs)
        nick = result.split(", ")[0].lower().replace(" ", "_")
        phrase = f"С гласа на {self._func.__name__.replace('_', ' ').title()}: {result}"
        if phrase not in self.MEMORY[nick]:
            self.MEMORY[nick].append(phrase)
        return result

memnick = Memnick
```

Реконструиране

- `__reduce__(self)`
- `__reduce_ex__(self, protocol)`

Method resolution order

Редът за обхождане на базови класове

```
class A(int): pass

class B: pass

class C(A, B, int): pass

C.__mro__ # или C.mro()
# <class '__main__.C'>, <class '__main__.A'>, <class '__main__.B'>,
# <class 'int'>, <class 'object'>
```

Method resolution order (2)

- Използва алгоритъм наречен C3 linearization
- https://en.wikipedia.org/wiki/C3_linearization
- https://dl.acm.org/doi/pdf/10.1145/236337.236343

Други (без конкретен ред)

- `__sizeof__`
- `__weakrefoffset__`
- `__base__`
- `__bases__`
- `__basicsize__`
- `__globals__`
- `__module__`
- `__package__`
- `__all__`
- `__builtins__`
- `__subclasses__`
- `__subclasshook__`
- `__subclasscheck__`
- `__file__`
- `__format__`
- `__loader__`
- `Ellipsis/...`

Код или данни?

```
def print_song(artist, song, album = None):
    result = f'The song "{song}"'
    if album:
        result += f' from the album "{album}"'
    result += f' is by {artist}'
    print(result)

print_song("David Bowie", "Earthling", "I'm Afraid of Americans")
print_song("Black Sabbath", "Black Sabbath", "Black Sabbath") # true story
```

Код или данни? (2)

```
>>> dir(print_song)
['__annotations__', '__call__', '__class__', '__closure__', '__code__',
'__defaults__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__',
'__format__', '__ge__', '__get__', '__getattribute__', '__globals__',
'__gt__', '__hash__', '__init__', '__init_subclass__', '__kwdefaults__',
'__le__', '__lt__', '__module__', '__name__', '__ne__', '__new__',
'__qualname__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__',
'__sizeof__', '__str__', '__subclasshook__']
>>> print_song.__name__
'print_song'
>>> print_song.__class__
<class 'function'>
>>> print_song.__defaults__
(None,)
```

Код или данни? (3)

```
>>> repr(print_song.__code__)
'<code object print_song at 0x000001C0DAA6C7A0, file "<pyshell#51>", line 1>'
>>> print_song.__code__.co_argcount # брой аргументи
3
>>> print_song.__code__.co_filename
'<stdin>'
>>> print_song.__code__.co_firstlineno
1
>>> print_song.__code__.co_stacksize
4
>>> print_song.__code__.co_names
('print',)
>>> print_song.__code__.co_code
b'\x97\x00d\x01|\x01\x9b\x00d\x02\x9d\x03}\x03|\x02r\t|\x03d\x03|\x02\x9b\x00d
\x02\x9d\x03z\r\x00\x00}\x03|\x03d\x04|\x00\x9b\x00\x9d\x02z\r\x00\x00}\x03t\x
01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00|\x03\xa6\x01\x00\x00\xab\x01\x00\x0
0\x00\x00\x00\x00\x00\x00\x01\x00d\x00S\x00'
```

ДИС 1

```
>>> dis.dis(print_song.__code__.co_code)
          0 RESUME                   0
          2 LOAD_CONST               1
          4 LOAD_FAST                1
          6 FORMAT_VALUE             0
          8 LOAD_CONST               2
         10 BUILD_STRING             3
         12 STORE_FAST               3
         14 LOAD_FAST                2
         16 POP_JUMP_FORWARD_IF_FALSE     9 (to 36)
         18 LOAD_FAST                3
         20 LOAD_CONST               3
         22 LOAD_FAST                2
         24 FORMAT_VALUE             0
         26 LOAD_CONST               2
         28 BUILD_STRING             3
         30 BINARY_OP               13 (+=)
         34 STORE_FAST               3
    >>   36 LOAD_FAST                3
         38 LOAD_CONST               4
         40 LOAD_FAST                0
         42 FORMAT_VALUE             0
         44 BUILD_STRING             2
         46 BINARY_OP               13 (+=)
         50 STORE_FAST               3
         52 LOAD_GLOBAL              1
         64 LOAD_FAST                3
         66 PRECALL                  1
         70 CALL                     1
         80 POP_TOP
         82 LOAD_CONST               0
         84 RETURN_VALUE
```

ДИС 2

```
>>> import dis
>>> dis.dis(print_song)
  1           0 RESUME                   0

  2           2 LOAD_CONST               1 ('The song "')
              4 LOAD_FAST                1 (song)
              6 FORMAT_VALUE             0
              8 LOAD_CONST               2 ('"')
             10 BUILD_STRING             3
             12 STORE_FAST               3 (result)

  3          14 LOAD_FAST                2 (album)
             16 POP_JUMP_FORWARD_IF_FALSE     9 (to 36)

  4          18 LOAD_FAST                3 (result)
             20 LOAD_CONST               3 (' from the album "')
             22 LOAD_FAST                2 (album)
             24 FORMAT_VALUE             0
             26 LOAD_CONST               2 ('"')
```

ДИС 3

```
              28 BUILD_STRING            3
              30 BINARY_OP              13 (+=)
              34 STORE_FAST              3 (result)

5       >>    36 LOAD_FAST               3 (result)
              38 LOAD_CONST              4 (' is by ')
              40 LOAD_FAST               0 (artist)
              42 FORMAT_VALUE            0
              44 BUILD_STRING            2
              46 BINARY_OP              13 (+=)
              50 STORE_FAST              3 (result)

6             52 LOAD_GLOBAL             1 (NULL + print)
              64 LOAD_FAST               3 (result)
              66 PRECALL                 1
              70 CALL                    1
              80 POP_TOP
              82 LOAD_CONST              0 (None)
              84 RETURN_VALUE
```

Инструкции

- Можем да произведем **изцяло** нов код
- Можем да сглобяваме нов at runtime
- За упражнение вкъщи!

Въпроси?