

```
In [1]: from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

Python 3

Классы - продолжение. Контекстные менеджеры

МИПТ 2020

Методы `__str__` и `__repr__`

```
In [2]: class Error:
        def __init__(self, err_string: str) -> None:
            self.err_string = err_string

        def __repr__(self) -> str:
            return f"Error(err_string='{self.err_string}')"

        def __eq__(self, other: 'Error') -> bool:
            return self.err_string == other.err_string

        def __str__(self) -> str:
            return self.err_string
```

```
In [3]: repr(Error(err_string='help me'))
str(Error(err_string='help me'))
Error(err_string='help me')
Error(err_string='help me') == Error(err_string='help me')
```

```
Out[3]: "Error(err_string='help me')"
```

```
Out[3]: 'help me'
```

```
Out[3]: Error(err_string='help me')
```

```
Out[3]: True
```

По умолчанию равенство работает как равенство ссылок на объекты, `__str__` вызывает `__repr__`

`__repr__` обычно отвечает за то, какие поля в данный момент лежат в классе

`__str__` дает представление класса как строки

Часто бывает так, что эти методы совпадают, если необычного `__str__` не предполагается

Dataclass (python3.7)

```
In [4]: import dataclasses

@dataclasses.dataclass
class ErrorData:
    err_string: str
```

Аннотация типа обязательна!

```
In [5]: repr(ErrorData(err_string='help me'))
str(ErrorData(err_string='help me'))
ErrorData(err_string='help me')
ErrorData(err_string='help me') == ErrorData(err_string='help me')
```

```
Out[5]: "ErrorData(err_string='help me')"
```

```
Out[5]: "ErrorData(err_string='help me')"
```

```
Out[5]: ErrorData(err_string='help me')
```

```
Out[5]: True
```

По дефолту декоратор `dataclass` сам создает функции `__init__`, `__repr__`, `__eq__` как в примере выше.

Можно передать в него аргументы и отключить функциональность

Арифметика

```
In [6]: import functools
from typing import Any

@dataclasses.dataclass
class Point:
    x: float
    y: float

    @functools.singledispatchmethod
    def __add__(self, other: 'Point') -> 'Point':
        return Point(x=self.x + other.x, y=self.y + other.y)

    @__add__.register
    def _(self, other: int) -> Any:
        return Point(x=self.x + other, y=self.y + other)

    def __radd__(self, other: int) -> 'Point':
        return Point(x=self.x + other, y=self.y + other)

    def __iadd__(self, other: 'Point') -> 'Point':
        self.x += other.x
        self.y += other.y
        return self

    def __mul__(self, other: 'Point') -> 'Point':
        return Point(x=self.x * other.x, y=self.y * other.y)

    def __imul__(self, other: 'Point') -> 'Point':
        self.x *= other.x
        self.y *= other.y
        return self
```

```
In [7]: pt_a = Point(x=2, y=3)
        pt_b = Point(x=2.5, y=-3.0)

        pt_a + pt_b
        pt_a + 2
        3 + pt_a
```

Out[7]: Point(x=4.5, y=0.0)

Out[7]: Point(x=4, y=5)

Out[7]: Point(x=5, y=6)

total_ordering

Этот декоратор позволяет определить все арифметические операции автоматически, если заданы `__eq__` и `__lt__`

```
In [8]: @functools.total_ordering
        @dataclasses.dataclass(eq=False)
        class KeyData:
            key: int
            data: str

            # Важно не забыть определить самостоятельно
            # Иначе равенство будет по паре (key, data)
            def __eq__(self, other: 'KeyData') -> bool:
                return self.key == other.key

            def __lt__(self, other: 'KeyData') -> bool:
                return self.key < other.key
```

```
In [9]: kd_a = KeyData(key=5, data='abc')
        kd_b = KeyData(key=3, data='bcd')
        kd_c = KeyData(key=3, data='ddd')

        kd_a < kd_b, 'lt'
        kd_a == kd_b, 'eq'
        kd_a <= kd_b, 'leq' # Автогенерированные
        kd_a > kd_b, 'gt'   # Автогенерированные

        kd_c == kd_b
```

Out[9]: (False, 'lt')

Out[9]: (False, 'eq')

Out[9]: (False, 'leq')

Out[9]: (True, 'gt')

Out[9]: True

Операторы контейнеров

<code>MutableMapping</code>	<code>Mapping</code>	<code>__getitem__</code> , <code>__setitem__</code> , <code>__delitem__</code> , <code>__iter__</code> , <code>__len__</code>	Inherited <code>Mapping</code> methods and <code>pop</code> , <code>popitem</code> , <code>clear</code> , <code>update</code> , and <code>setdefault</code>
-----------------------------	----------------------	---	---

```
In [10]: from collections.abc import MutableMapping
from typing import Dict, Iterator

@dataclasses.dataclass
class XorEncodedDict(MutableMapping):
    encode_key: int
    dct: Dict[int, int] = dataclasses.field(default_factory=dict)

    def __getitem__(self, key: int) -> int:
        return self.dct[key] ^ self.encode_key

    def __setitem__(self, key: int, value: int) -> None:
        self.dct[key] = value

    def __delitem__(self, key: int) -> int:
        return self.dct.pop(key)

    def __iter__(self) -> Iterator[int]:
        return iter(self.dct)

    def __len__(self) -> int:
        return len(self.dct)
```

```
In [11]: enc = XorEncodedDict(encode_key=15)
enc[10] = 1
enc[5] = 13

print(enc)

enc.setdefault(5, 0)

enc.popitem()

print("items:")

for k, v in enc.items():
    k, v
```

```
XorEncodedDict(encode_key=15, dct={10: 1, 5: 13})
```

```
Out[11]: 2
```

```
Out[11]: (10, 14)
```

```
items:
```

```
Out[11]: (5, 2)
```

Контекстные менеджеры

Базовый пример - открытие файла

```
In [12]: with open('out_file.txt', 'w') as out:
out.write('Hello world!')

with open('out_file.txt') as inp:
    inp.read()
```

```
Out[12]: 12
```

```
Out[12]: 'Hello world!'
```

Можно написать свой аналог для файлов

In [13]: **from typing import Any**

```
@dataclasses.dataclass
class MyOpener:
    file_path: str

    def __enter__(self) -> Any:
        self.file = open(self.file_path)
        return self.file

    def __exit__(self, exc, exc_info, exc_trace) -> None:
        print(exc, exc_info, exc_trace)
        self.file.close()
        print('File is closed!')
```

In [14]: **try:**
 with MyOpener('out_file.txt') **as** inp:
 inp.read()
 raise Exception("Some exception happened")
except Exception:
 pass

Out[14]: 'Hello world!'

```
<class 'Exception'> Some exception happened <traceback object at 0x7fe11c34df40>
>
File is closed!
```

Файл закрылся, несмотря на исключение, это хорошо

Есть более простой способ создания контекстных менеджеров

In [15]: **import** contextlib

```
@contextlib.contextmanager
def MyCtxlibOpener(file_path: str) -> Any:
    file = open(file_path)
    try:
        yield file
    finally:
        file.close()
        print('File is closed!')
```

In [16]: **try:**
 with MyCtxlibOpener('out_file.txt') **as** inp:
 inp.read()
 raise Exception("Some exception happened")
except Exception:
 pass

Out[16]: 'Hello world!'

File is closed!

Метод `__new__`

Вызывается до `__init__` и решает, как создавать объект

```
In [17]: import random
from typing import Optional

class CheshireCat:
    def __init__(self) -> None:
        print("Miau, I'm here")

    def __new__(cls) -> Optional['CheshireCat']:
        if random.randint(0, 1) == 1:
            return object.__new__(cls)
        else:
            print("Anyone saw a cat?")
            return None
```

```
In [18]: for i in range(5):
        cat = CheshireCat()

cat
```

```
Anyone saw a cat?
Miau, I'm here
Anyone saw a cat?
Miau, I'm here
Miau, I'm here
```

```
Out[18]: <__main__.CheshireCat at 0x7fe11cb87c10>
```

Пример с наследованием

```
In [19]: class CheshireKittenMolly(CheshireCat):
        def __init__(self) -> None:
            super().__init__()
            print("I'm Molly!")

        def __new__(cls) -> Optional['CheshireKittenMolly']:
            if not hasattr(cls, 'instance'):
                cls.instance = None
                while cls.instance is None:
                    cls.instance = super(CheshireCat, cls).__new__(cls)
                    cls.instance.milk_volume = 5
                    cls.instance.name = 'Molly'
            return cls.instance
```

```
In [20]: kitten1 = CheshireKittenMolly()
        kitten1.milk_volume = 3

        kitten2 = CheshireKittenMolly()
        kitten2.milk_volume

        id(kitten1), id(kitten2)
        kitten1 is kitten2
```

```
Miau, I'm here
I'm Molly!
Miau, I'm here
I'm Molly!
```

```
Out[20]: 3
```

```
Out[20]: (140604826084976, 140604826084976)
```

```
Out[20]: True
```

