Exceptions

Alexander Evgin

21 февраля 2020 г.

Outline

Recap

- ООП (инкапсуляция, наследование, полиморфизм)
- Объект (**id**, **type**, value)
- Класс
- Атрибут экземпляра класса (объекта), атрибут класса

```
1   class A:
2       x = 0
3
4       def __init__(self):
5           self.y = 1
6
```

- Magic-методы (`__str__`, `__eq__`, `__getitem__`, ...)

Deeper into classes

```
1   class A:
2       pass
3
```

Deeper into classes

```
1   class A:
2       y = 1
3
4       def __init__(self):
5           self.x = 0
6
7       def get_x(self):
8           return self.x
9
10  a = A()
11
```

Deeper into classes

```
1   class A:
2       y = 1
3
4       def __init__(self):
5           self.x = 0
6
7       def get_x(self):
8           return self.x
9
10  a = A()
11
```

```
>>> type(A.get_x)
<class 'function'>
>>> type(a.get_x)
<class 'method'>
```

@classmethod

```
1   class A:
2       y = 1
3
4       @classmethod
5       def get_y(cls):
6           return cls.y
7
8   a = A()
9
```

```
>>> type(A.get_y)
<class 'method'>
>>>
```

@classmethod

```
1   class MyClass:
2       def __init__(self, param1, param2):
3           self.x = param1
4           self.y = param2
5
6       @classmethod
7       def new(cls): # or new(cls, params, ...)
8
9           # preparing some stuff
10          param1 = 1.
11          param2 = 2.
12
13          return cls(param1, param2)
14
15
16  inst_1 = MyClass.new()
17
```

@staticmethod

```
1   class RequestHandler:
2       def other_method(self, addr):
3           addr = self.unwrap_address(addr)
4           ...
5
6       @staticmethod
7       def unwrap_address(addr):
8           unwrapped = [b if b is not None else 0
9                        for b in addr]
10          return unwrapped
11
```

__slots__

```
>>> class A:
...     __slots__ = ["x", "y"]  # экономим память
...
>>> a = A()
>>> a.__dict__
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'A' object has no attribute '__dict__'
>>> a.x = 92
>>> a.z = 92
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'A' object has no attribute 'z'
```

`__hash__`

Hashable type

- `a == b` $\rightarrow$ `hash(a) == hash(b)` (pay attention to `__eq__`)
- hashable objects can be keys in dict

Наследование

```
class Counter:
    def __init__(self, initial_count=0):
        self.count = initial_count

    def get(self):
        return self.count


class SquaredCounter(Counter):
    def get(self):
        return super().get() ** 2


c = SquaredCounter(91)
assert c.get() == 8281
```

Наследование

```
class A:
    def f(self):
        print("A")

class B:
    def f(self):
        print("B")

class C(A, B):
    pass

C().f()
# A
```

```
class Base:
    def f(self):
        print("Base")

class A(Base):
    def f(self):
        print("A")
        super().f()  # super is dynamic!

class B(Base):
    def f(self):
        print("B")
        super().f()

class C(A, B):
    pass

C().f()
# A
# B
# Base

assert C.mro() == [C, A, B, Base, object]
```

Mixin

```
class DoublingMixing:  # !!!
    def increment(self):
        super().increment()
        super().increment()


class DoublingCounter(DoublingMixing, Counter):
    pass


c = DoublingCounter()
assert c.count == 0
c.increment()
assert c.count == 2
```

Inheritance

`object` — базовый класс любого класса

```
>>> class A:
...   pass
...
>>> dir(A)
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__
', '__eq__', '__format__', '__ge__', '__getattribute__', '__
gt__', '__hash__', '__init__', '__init_subclass__', '__le__'
, '__lt__', '__module__', '__ne__', '__new__', '__reduce__',
 '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '
__str__', '__subclasshook__', '__weakref__']
>>>
```

Exceptions

Exceptions

```
1   cars = {'audi': 10, 'mercedes-benz': 17, 'toyota': 24}
2
3   ford = cars['ford']
4
```

```
Traceback (most recent call last):
  File "dict_failure.py", line 3, in <module>
    ford = cars['ford']
KeyError: 'ford'
```

```
1   a = 1 / 0
2
```

```
Traceback (most recent call last):
  File "div_failure.py", line 1, in <module>
    a = 1 / 0
ZeroDivisionError: division by zero
```

Operator `raise`

```python
def division(first: float, second: float):
    if second == 0:
        raise ZeroDivisionError
    return first / second


division(15, 0)

```

```
Traceback (most recent call last):
  File "division.py", line 6, in <module>
    division(15, 0)
  File "division.py", line 3, in division
    raise ZeroDivisionError
ZeroDivisionError
```

Operator `raise`

Более информативно:

```
1   def division(first: float, second: float):
2       if second == 0:
3           raise ZeroDivisionError('Do not divide by zero!')
4       return first / second
5
6
7   division(15, 0)
8
```

```
Traceback (most recent call last):
  File "division.py", line 7, in <module>
    division(15, 0)
  File "division.py", line 3, in division
    raise ZeroDivisionError('Do not divide by zero!')
ZeroDivisionError: Do not divide by zero!
```

Exception handling syntax

```
try:
    ...  # run some dangerous stuff
except (...): # type of exceptions specified here
    ... # run if exception catched
else:
    ... # run if no exception
finally:
    ... # run anyway
```

Exception handling syntax

```python
def division(fist: float, second: float):
    if second == 0:
        raise ZeroDivisionError('Zero division attempt')
    return fist / second

try:
    a = division(15, 0)
except ZeroDivisionError as e:
    print('Something went wrong: {}'.format(e))
else:
    print('Succeed!')
finally:
    print('(You will see this anyway)')
```

```
Something went wrong: Zero division attempt
(You will see this anyway)
```

Built-in exceptions hierarchy

```
BaseException
 +-- SystemExit
 +-- KeyboardInterrupt
 +-- GeneratorExit
 +-- Exception
      +-- StopIteration
      +-- StopAsyncIteration
      +-- ArithmeticError
      |    +-- FloatingPointError
      |    +-- OverflowError
      |    +-- ZeroDivisionError
      +-- AssertionError
      +-- AttributeError
      +-- BufferError
      +-- EOFError
      +-- ImportError
      |    +-- ModuleNotFoundError
      +-- LookupError
      |    +-- IndexError
      |    +-- KeyError
      +-- MemoryError
      +-- NameError
      |    +-- UnboundLocalError
      +-- OSError
      |    +-- BlockingIOError
      |    +-- ChildProcessError
      |    +-- ConnectionError
      |    |    +-- BrokenPipeError
      |    |    +-- ConnectionAbortedError
      |    |    +-- ConnectionRefusedError
      |    |    +-- ConnectionResetError
      |    +-- FileExistsError
      |    +-- FileNotFoundError
      |    +-- InterruptedError
      |    +-- IsADirectoryError
      |    +-- NotADirectoryError
      |    +-- PermissionError
      |    +-- ProcessLookupError
      |    +-- TimeoutError
```

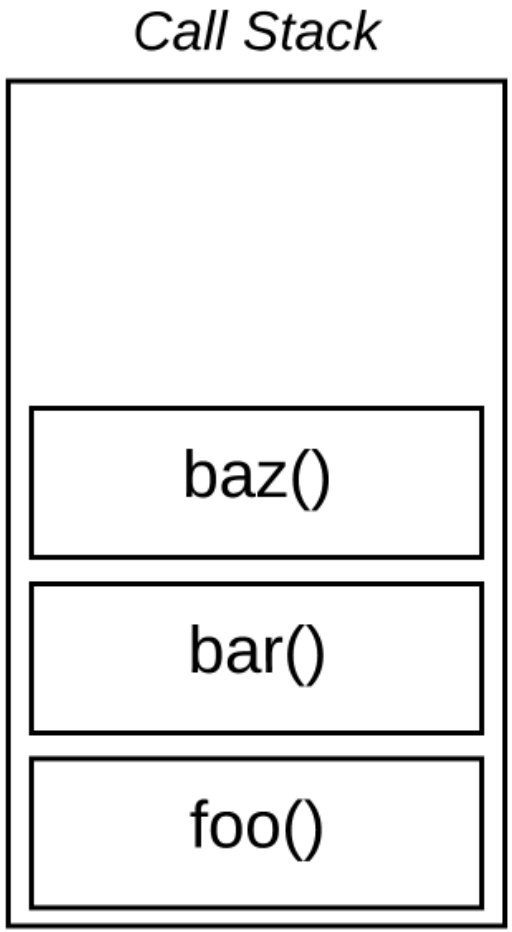
Traceback

```
1   def outer():
2       try:
3           middle()
4       except Exception as e:
5           print("Exception: {}".format(e))
6           raise e
7
8   def middle():
9       try:
10          inner()
11      finally:
12          print("cleanup")
13
14  def inner():
15      raise RuntimeError("Kaboom")
16
17  outer()
```

```
cleanup
Exception: Kaboom
Traceback (most recent call last):
  File "traceback_ex.py", line 17, in <module>
    outer()
  File "traceback_ex.py", line 6, in outer
    raise e
  File "traceback_ex.py", line 3, in outer
    middle()
  File "traceback_ex.py", line 10, in middle
    inner()
  File "traceback_ex.py", line 15, in inner
    raise RuntimeError("Kaboom")
RuntimeError: Kaboom
```

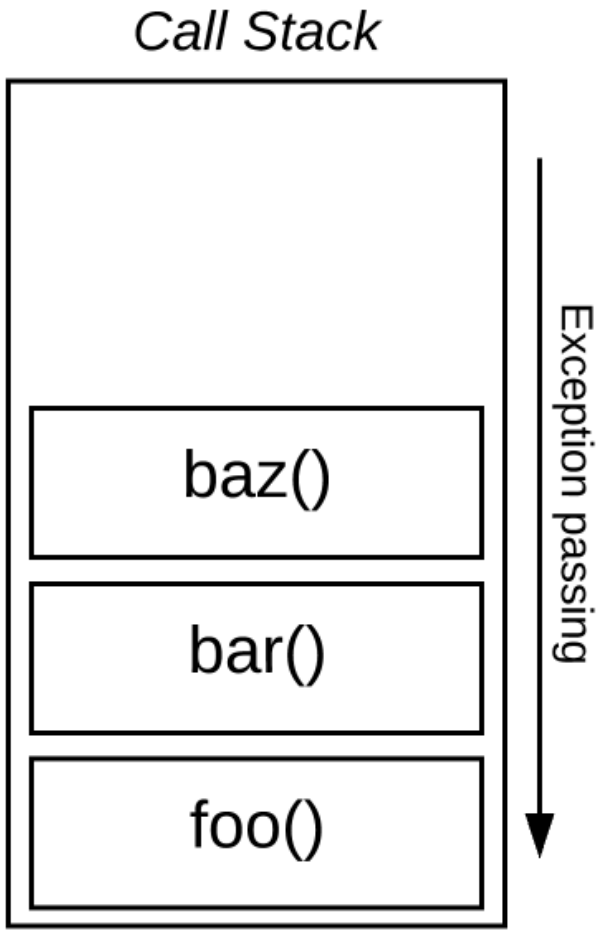
Call stack

```
1   def foo():
2       bar()
3
4   def bar():
5       baz()
6
7   def baz():
8       pass
9
10  foo()
```



Call stack

```
1   def foo():
2       bar()
3
4   def bar():
5       baz()
6
7   def baz():
8       pass
9
10  foo()
```



RuntimeError

```
>>> d = {"foo": 42, "bar": 24}
>>> for key in d:
...     d.pop(key)
...
42
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
RuntimeError: dictionary changed size
    during iteration
```

ImportError

```
try:
    import foobar_speedups as foobar
except ImportError:
    import foobar_slow as foobar
```

AttributeError

```
>>> class A:
...     pass
...
>>> A().foobar
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'A' object has no attribute 'foobar'
```

AttributeError

```
>>> class A:
...     pass
...
>>> A().foobar
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'A' object has no attribute 'foobar'
```

Может ли AttributeError возникнуть при записи?

LookupError

```
>>> [][0]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
>>> {}[0]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 0
```

TypeError

```
>>> [][None]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: list indices must be integers or slices,
    not NoneType
```

ValueError

```
>>> int("XXI")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10:
    'XXI'
```

Собственные исключения

```
class Error(Exception):
    """Exception that is the base class of all
    other error exceptions.
    You can use this to catch all errors with
    one single except statement.
    """
    pass

class DatabaseError(Error):
    """Exception that are related to the database.
    """
    pass

class InterfaceError(Error):
    ...
```

Custom exceptions

`raise` “выбрасывает” объект! Исключения — объекты!

API Исключений

```
>>> e = Exception("hello", 92, "world")
>>> e.args
('hello', 92, 'world')
>>> raise e
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
Exception: ('hello', 92, 'world')
>>> e.__traceback__
<traceback object at 0x7efcae9dec48>
```

API Исключений

```
>>> e2 = Exception()
>>> e2.__traceback__ is None
True
>>> e3 = e2.with_traceback(e.__traceback__)
>>> e3 is e2 and e3.__traceback__ is not None
True
```

```python
import traceback

def foo():
    bar()

def bar():
    raise Exception  # raise Exception()

try:
    foo()
except Exception as e:
    traceback.print_tb(e.__traceback__)

# File "main.py", line 11, in <module>
#   foo()
# File "main.py", line 4, in foo
#   bar()
# File "main.py", line 7, in bar
#   raise Exception <- причина внизу
```

Исключения -- причины

```
class LibraryError(Exception):
    pass

try:
    open("I_don't_exist.rly")
except OSError:
    raise LibraryError
```

Исключения -- причины

```
Traceback (most recent call last):
  File "main.py", line 5, in <module>
    open("I_don't_exist.rly")
FileNotFoundError: [Errno 2] No such file or directory:
    "I_don't_exist.rly"

During handling of the above exception,
    another exception occurred:

Traceback (most recent call last):
  File "main.py", line 7, in <module>
    raise LibraryError
__main__.LibraryError
```

e.__context__ -- исключение-контекст

Исключения -- причины

```
class LibraryError(Exception):
    pass


try:
    open("I_don't_exist.rly")
except OSError as e:
    raise LibraryError from e
```

Исключения -- причины

```
Traceback (most recent call last):
  File "main.py", line 6, in <module>
    open("I_don't_exist.rly")
FileNotFoundError: [Errno 2] No such file or directory:
    "I_don't_exist.rly"

The above exception was the direct cause
    of the following exception:

Traceback (most recent call last):
  File "main.py", line 8, in <module>
    raise LibraryError from e
__main__.LibraryError
```

e.__cause__ -- исключение-причина

Исключения -- причины

```
class LibraryError(Exception):
    pass


try:
    open("I_don't_exist.rly")
except OSError as e:
    raise LibraryError from None

# Traceback (most recent call last):
#   File "main.py", line 8, in <module>
#     raise LibraryError from None
# __main__.LibraryError
```

Исключения -- причины

```
try:
    open("I_don't_exist.rly")
finally:
    open("log.txt")
```

raise

```
raise Exception("foo")
raise Exception("foo") from e
raise Exception("foo") from None
raise # re-raises last exception
```

Q & A

Задание

- Реализовать класс `Graph`:
  - узлы — объекты класса `Node` (имеют уникальные `id`)
  - представление графа с помощью списков смежности
  - реализация "необходимых методов" (например, `in_graph(node)` или `add_node(node)`)
- Реализовать класс `WeightedGraph`, наследник `Graph`:
  - хранение весов рёбер

Самостоятельное задание

Дополнительно к основному заданию

- ***(5 баллов)*** Реализовать обход графа в ширину: метод `bfs(start_node)` $\rightarrow$ `list`: (возвращает список узлов в порядке посещения)
- ***(15 баллов)*** Реализовать метод `is_tree()`, проверяющий, является ли граф деревом (возвращает `True` или `False`)