Python2018

compscicenter.ru

aleksey.kladov@gmail.com

Лекция 7

Работа с исключениями

try/except/finally

```python
answer = 92
try:
    if answer != 42:
        raise ValueError("Wrong answer")
except RuntimeError as e:
    print(f"Error occurred {e}")
    raise e
finally:
    print("Cleanup")
```

try/except/finally

```python
def outer():
    try:
        middle()
    except Exception as e:
        print("Exception: {e}")
        raise e


def middle():
    try:
        inner()
    finally:
        print("cleanup")


def inner():
    raise RuntimeError("Kaboom")
```

Философия

almost all (92%) of the catastrophic system failures are the result of incorrect handling of non-fatal errors explicitly signaled in software.

https://www.usenix.org/system/files/conference/osdi14/osdi14-paper-yuan.pdf

Философия

- обработка ошибок редко тестируется
- обработчики ошибок редко срабатывают в production

Backup problem

- Можно восстанавливать backup каждую ночь
- crash only software -- нет кнопки "штатного" завершения

Виды ошибок

- Не ошибки: d.get(key)
- Ошибки: d[key]

Виды ошибок

- open("kittens.jpeg")
- ошибка для command line утилиты
- ожидаемый случай для графического редактора

Типичная обработка ошибок

```
setup_resources()
try:
    may_fail()
finally:
    cleanup_resources()
```

Изоляция ошибок

```
for request in user_requests:
    try:
        state = handle_request(state, request)
    except Exception as e:
        show_error_dialog(e)
```

Типичные границы

- процесс
- поток
- запрос пользователя

Обратно к Python

```
try:
    something_dangerous()
except (ValueError, ArithmeticError): # any of
    pass
except TypeError as e:  # isinstance(e, TypeError)
    pass
```

Иерархия исключений

```
>>> BaseException.__subclasses__()
[<class 'Exception'>,
 <class 'GeneratorExit'>,
 <class 'SystemExit'>,
 <class 'KeyboardInterrupt'>]
>>> len(Exception.__subclasses__())
19
>>> Exception.__subclasses__()[:5]
[<class 'TypeError'>,
 <class 'StopAsyncIteration'>,
 <class 'StopIteration'>,
 <class 'ImportError'>,
 <class 'OSError'>]
```

Поймать всё

```
import sys

try:
    sys.exit()
except:  # плохо
    pass

try:
    sys.exit()
except:  # плохо
    pass

try:
    sys.exit()
except Exception:  # catch all
    pass
```

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):
    ...
```

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
```

else

```
try:
    file = open("example.txt", "w")
except IOError as e:
    print(e, file=sys.stderr)
else:
    report_success(file)
```

else

```
try:
    file = open("example.txt", "w")
except IOError as e:
    print(e, file=sys.stderr)
else:
    report_success(file)

# ---------------------------------- #

try:
    file = open("example.txt", "w")
    report_success(file)
except IOError as e:
    print(e, file=sys.stderr)
```

Типичная обработка ошибок

```
db = open("database.txt")
try:
    write_to_database(db)
finally:
    db.close()
```

Типичная обработка ошибок

```python
db = open("database.db")
db2 = open("database.db2")  # ooups
try:
    write_to_database(db, db2)
finally:
    db.close()
    db2.close()  # ooups
```

with

```
with open("database.db") as db:
    with open("database.db2") as db2:
        write_to_database(db, db2)
    # calls db2.close() automatically
# calls db.close() automatically
```

with

```python
with open("database.db") as db, \
     open("database.db2") as db2:
    write_to_database(db, db2)
```

with

```python
with open("input.txt") as f:
    text = f.read()
    process(text)
```

with

```
with open("input.txt") as f:
    text = f.read()

process(text)  # 👍
```

with

```python
with acquire_resource() as r:
    do_something(r)
```

with

```
# Не правда
manager = acquire_resource()
r = manager.__enter__()
try:
    do_something(r)
finally:
    manager.__exit__()
```

with

```
manager = acquire_resource()
r = manager.__enter__()
try:
    do_something(r)
finally:
    # None, None, None если исключения нет
    exc_type, exc_value, tb = sys.exc_info()
    suppress = manager.__exit__(exc_type, exc_value, tb)
    if exc_value is not None and not suppress:
        raise exc_value
```

```python
from functools import partial

class opened:
    def __init__(self, path, *args, **kwargs):
        self.opener = partial(open, path, *args, **kwargs)

    def __enter__(self):
        self.file = self.opener()
        return self.file

    def __exit__(self, *exc_info):
        self.file.close()
        del self.file
```

```python
manager = opened("foo.txt")
with manager as f:
    with manager as g:
        pass
    # закрыли второй дескриптор
# первый дескриптор открыт
```

```python
from contextlib import AbstractContextManager  # >= 3.6
from functools import partial

class opened(AbstractContextManager):
    def __init__(self, path, *args, **kwargs):
        self.opener = partial(open, path, *args, **kwargs)

    def __enter__(self):
        self.file = self.opener()
        return self.file

    def __exit__(self, exc_type, exc_value, traceback):
        self.file.close()
        del self.file
```

```
# Файлы -- менеджеры контекста

class IOBase:
    ...

    def __enter__(self):
        self._checkClosed()
        return self  # !

    def __exit__(self, *args):
        self.close()

    ...
```

```python
import os


class cd:
    def __init__(self, target):
        self.target = target

    def __enter__(self):
        self.saved_cwd = os.getcwd()
        os.chdir(self.target)

    def __exit__(self, *exc_info):
        os.chdir(self.saved_cwd)
        del self.saved_cwd


print(os.getcwd())      # /home/matklad/python-2018
with cd("/tmp"):
    print(os.getcwd()) # /tmp
print(os.getcwd())      # /home/matklad/python-2018
```

NamedTemporaryFile

```python
import tempfile

with tempfile.NamedTemporaryFile() as file:
    path = file.name
    assert path.startswith("/tmp")
```

contextlib

```python
import io
from contextlib import redirect_stdout

file = io.StringIO()
with redirect_stdout(file):
    print("Hello, world!")


assert file.getvalue() == "Hello, world!\n"
```

contextlib

```python
import os
from contextlib import suppress

with suppress(FileNotFoundError):
    os.remove("non-existing-file.txt")
```

contextlib

```
class supress:
    def __init__(self, *suppressed):
        self.suppressed = suppressed

    def __enter__(self):
        pass

    def __exit__(self, exc_type, exc_value, tb):
        return (exc_type is not None and
                issubclass(exc_type, suppressed))
```

contextlib

```python
from contextlib import suppress, ContextDecorator


class suppressed(suppress, ContextDecorator):
    pass


@suppressed(IOError)
def do_something():
    ...
```

Почитать в транспорте

http://joeduffyblog.com/2016/02/07/the-error-model/