Python (BSU FAMCS Fall'19)

Семинар 4

Преподаватель: Дмитрий Косицин

Задание 1. (1 балл). Реализуйте декоратор `handle_error`, который позволяет обрабатывать и логировать ошибки в зависимости от переданных параметров:

- `re_raise` – флаг, отвечающий за то, будет произведен проброс исключения типа `exc_type` из блока/функции на уровень выше или нет (по умолчанию `True`, исключения типов, не указанных или не наследующихся от `exc_type`, должны пробрасываться выше по стеку безусловно);
- `log_traceback` – флаг, отвечающий за то, будет ли при возникновении исключения типа `exc_type` отображен `traceback` (по умолчанию `True`);
- `exc_type` – параметр, принимающий либо отдельный тип, либо непустой кортеж типов исключений, которые должны быть обработаны (для всех остальных блока `except` не будет) – значение по умолчанию выставьте тип `Exception`;
- `tries` – параметр, означающий количество попыток вызова функции, прежде чем бросить исключение (по умолчанию `1`, значение `None` – бесконечные попытки, неположительные значения недопустимы);
- `delay` – значение задержки между попытками в секундах (может быть `float`, по умолчанию 0);
- `backoff` – значение множителя, на который умножается `delay` с каждой попыткой (по умолчанию 1, см. пример)

Логирование можно осуществлять с помощью глобального для модуля объекта `logger` – `Logger`'а из стандартной библиотеки (модуль `logging`).

Сохраните декоратор в файле *error_handling.py*.

Пример 1

```
# suppress exception, log traceback
@handle_error(re_raise=False)
def some_function():
    x = 1 / 0  # ZeroDivisionError

some_function()
print(1)  # line will be executed as exception is suppressed
```

Пример 2

```
# re-raise exception and doesn't log traceback as exc_type doesn't match
@handle_error(re_raise=False, exc_type=KeyError)
def some_function():
    x = 1 / 0  # ZeroDivisionError

some_function()
print(1)  # line won't be executed as exception is re-raised
```

Пример 3

Пусть в примере ниже `random.random()` последовательно возвращает 0.2, 0.5, 0.3, тогда декоратор должен вызвать функцию `some_function`, перехватить исключение, подождать 0.5 секунды, попробовать еще раз, подождать 1 секунду, попробовать еще раз и пробросить исключение.

```python
import random

@handle_error(re_raise=True, tries=3, delay=0.5, backoff=2)
def some_function():
    if random.random() < 0.75:
        x = 1 / 0  # ZeroDivisionError

some_function()
```

Задание 2. (0.5 балла). Реализуйте контекстный менеджер `handle_error_context`, аналогичный декоратору `handle_error`, который позволяет обрабатывать и логировать ошибки в зависимости от переданных параметров:

- `re_raise` – флаг, отвечающий за то, будет произведен проброс исключения типа `exc_type` из блока/функции на уровень выше или нет (по умолчанию `True`, исключения типов, не указанных или не наследующихся от `exc_type`, должны пробрасываться выше по стеку безусловно);
- `log_traceback` – флаг, отвечающий за то, будет ли при возникновении исключения типа `exc_type` отображен `traceback` (по умолчанию `True`);
- `exc_type` – параметр, принимающий либо отдельный тип, либо непустой кортеж типов исключений, которые должны быть обработаны (для всех остальных блока `except` не будет) – значение по умолчанию выставьте тип `Exception`.

Обратите внимание, что при реализации менеджера контекста код предлагается переиспользовать, чтобы избежать дублирования кода. Реализовывать менеджер контекста с помощью класса также нежелательно.

Сохраните контекстный менеджер в файле *error_handling.py*.

Пример

```python
# log traceback, re-raise exception
with handle_error_context(log_traceback=True, exc_type=ValueError):
    raise ValueError()
```

Задание 3. (1 балл). Реализуйте метакласс `BoundedMeta`, который контролирует количество созданных объектов классов, которые имеют данный метакласс. Допустимое количество объектов задайте параметром (по умолчанию 1).

В случае превышения бросайте исключение `TypeError`. Если значение параметра равно `None`, то ограничений нету.

Другими словами, у класса `C` с метаклассом `BoundedMeta` должно быть создано не более 2 экземпляров.

Класс сохраните в файле *functional.py*.

Заготовка метакласса BoundedMeta

```python
class C(metaclass=BoundedMeta, max_instance_count=2):
    pass

c1 = C()
c2 = C()

try:
    c3 = C()
except TypeError:
    print('everything works fine!')
else:
    print('something goes wrong!')
```

Задание 4. (1 балл). Реализуйте класс `BoundedBase`, в котором определен абстрактный метод класса `get_max_instance_count`, возвращающий максимальное количество экзмепляров, которые можно создать.

Не допускайте создания объекта, если данное значение превышено – бросайте исключение `TypeError`. Значение, равное `None` – без ограничений.

Класс сохраните в файле *functional.py*.

Заготовка класса BoundedBase

```
class D(BoundedBase):
    @classmethod
    def get_max_instance_count(cls):
        return 1

d1 = D()

try:
    d2 = D()
except TypeError:
    print('everything works fine!')
else:
    print('something goes wrong!')
```

Задание 5. (0.5 балла). Напишите функцию, которая при каждом вызове возвращает количество раз, которое она была вызвана. Использовать глобальные переменные не допускается, иначе говоря, в файле должно быть только определение функции (с ключевым словом `def`).

Функцию назовите `smart_function` и сохраните в файле *functional.py*.

Пример

```
for real_call_count in range(1, 5):
  assert f() == real_call_count
```