Python (BSU FAMCS Fall'18)

Семинар 4

Преподаватели: Дмитрий Косицин, Светлана Боярович

Общие замечания ко всем заданиям

Свой тестирующий код можно размещать под условием `if __name__ == '__main__'` внизу файла. Такой код выполнится только если запустить этот файл, но не импортировать его.

Просьба использовать те имена для функций и файлов, которые указаны в замечаниях к заданиям.

Для отправки заданий выберите в anytask нужную задачу и там к сообщению прикрепите свое решение (один или несколько .py файлов).

Для тестирования интерфейса ваших заданий выложен специальный скрипт. Просьба также не оставлять debug print'ы в ваших программах, которые нужны исключительно для вывода отладочной информации.

Задание 1. (0.5 балла). Реализуйте декоратор `profile`, который при вызове функции подсчитывает время выполнения этой функции, выводит его на экран и возвращает результат вызова функции.

Рассмотрите стандартный модуль `timeit` и, в частности, функцию `default_timer` для измерения времени выполнения.

Декоратор сохраните в файле *utils.py*.

Пример

```
@profile
def some_function():
    return sum(range(1000))

result = some_function()  # return a value and print execution time
```

Задание 2. (0.5 балла). Реализуйте менеджер контекста `timer`, который посчитает время выполнения блока и выведет его на экран.

Сохраните менеджер контекста в файле *utils.py*.

Пример

```
with timer():
    print(sum(range(1000)))
    # print execution time when calculation is over
```

Задание 3. (1.5 балла). Реализуйте callable-класс `SafeRequest`, которой позволяет выполнить запрос к ресурсу, ограничив его по *timeout*, а также вернуть некоторое значение по умолчанию в случае получения кода ответа 404, если оно установлено. Если же значение по умолчанию не установлено, то при таком коде ответа возникшее исключение должно быть проброшено дальше.

В конструкторе класса передавайте два параметра:

- timeout – время таймаута в секундах, может быть `float`, по умолчанию 3;
- default – `None` или *not set* (см. паттерн в лекциях) – значение по умолчанию, если страница отсутствует.

Предполагайте, что запрос выполняется с помощью библиотеки `requests`. Для проверки значения `timeout` можно использовать аннотации типов (PEP-484).

Сохраните класс в файле *utils.py*.

Пример

```
import requests

# ordinary way; requests.get may be used
data = requests.request('get', 'http://yandex.ru/', headers={...})

# timeout added to request, default value might be used
safe_request = SafeRequest(timeout=5, default=None)
data = safe_request('get', 'http://yandex.ru/', headers={...})
```

Задание 4. (2.5 балла). Реализуйте контекстный менеджер `handle_error_context` и декоратор `handle_error`, которые позволяют обрабатывать и логировать ошибки в зависимости от переданных параметров:

- `re_raise` – флаг, отвечающий за то, будет произведен проброс исключения (типы исключений для обработки заданы параметром `exc_type` – см. ниже) из блока/функции на уровень выше или нет (по умолчанию `True`);
- `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)

Для обработки и логирования *traceback* можно использовать функцию `sys.exc_info()` и схожие ей, модуль `traceback`, а логирование осуществлять с помощью глобального для модуля объекта `logger` – `Logger`'а из стандартной библиотеки (модуль `logging`).

Обратите внимание, что при реализации декоратора и менеджера контекста код должен быть переиспользован – простого копирования требуется избежать. Реализовывать менеджер контекста с помощью класса также не нужно.

Сохраните все в файле *error_handling.py*.

Замечание. Применение аналога `SafeRequest` в связке с `retry` – хорошая практика при выполнении запросов.

Пример 1

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

Пример 2

```
# 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 3

```
# 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 4

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.

```
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()
```