Описание к домашней работе:

1. Файл «task6\_1.py»

Попытка проанализировать 4 варианта решения задания по «переворачиванию» числа с помощью декоратора @profile (2 варианта решены циклом, 2 – рекурсией).

Функция, реализуемая декоратором @profile, показала, что инкрементов ни в одном вызове не обнаружено, а изначально выделяемый объем памяти – одинаковый: 15,6-15.7 MiB. Это может означать, что:

1. Шаг измеряемого инкремента (MiB) существенно больше выделяемой в каждой строке кода памяти всех функций (это будет описано ниже, при анализе с помощью memory\_usage().
2. Память выделяется в целом на программу, и при вызове разных функций в рамках одной программы разницы в MiB не заметно.

В связи с этим возникла идея проанализировать по отдельности две функции – самую неэффективную (getreversenumber1(), через рекурсию) и оптимальную (getreversenumber4(), через цикл) в отдельных запусках (скриптах).

1. Файлы «task6\_1.1.py» и «task6\_1.4.py»

Функции getreversenumber1() (файл «task6\_1.1.py») и getreversenumber4() (файл «task6\_1.4.py») проанализированы по отдельности. Результаты профилирования идентичны: инкрементов нет, объем выделенной памяти – 15,5-15,6 MiB. С точки зрения эффективности выделения памяти существенной разницы между функциями нет – в пределах 1 MiB. При этом, напрямую в рекурсивных функциях использовать @profiler не удобно (вызывается столько же раз, сколько происходит рекурсивный вызов), поэтому предполагаю, что для профилирования рекурсивных функций целесообразно оборачивать их в обычную функцию, которая вызывает рекурсивную и декорируется. В таком случай накладных расходов будет минимум, а профилировка будет проведена только 1 раз.

1. Файл «task6\_1.4a.py»

Функция getreversenumber4() помимо профилирования проанализирована с помощью замеров memory\_usage() и guppy.heap().

Выводы:

@profile обнаружил бОльшую память, выделенную под программу по сравнению с файлами «task6\_1.py», «task6\_1.1.py» и «task6\_1.4.py»: 15,9-16,0 MiB против ~15,6 MiB. Очевидно, это связано с подключением дополнительных библиотек:

from memory\_profiler import memory\_usage  
from guppy import hpy

При этом инкрементов также не выявлено.

Разница используемой памяти до и после вызова анализированной функции составила 0.1640625 MB, что косвенно подтверждает несущественное значение для профилировщика @profile и отсутствие инкрементов.

Функция guppy.heap() выдала дамп используемых в коде структур.