Profiling All Versions

Examples

%%timeit and %timeit in IPython

Profiling string concatanation:

```
In [1]: import string

In [2]: %%timeit s=""; long_list=list(string.ascii_letters)*50
  ....: for substring in long_list:
  ....:   s+=substring
  ....:
1000 loops, best of 3: 570 us per loop

In [3]: %%timeit long_list=list(string.ascii_letters)*50
  ....: s="".join(long_list)
  ....:
100000 loops, best of 3: 16.1 us per loop
```

Profiling loops over iterables and lists:

```
In [4]: %timeit for i in range(100000):pass
100 loops, best of 3: 2.82 ms per loop

In [5]: %timeit for i in list(range(100000)):pass
100 loops, best of 3: 3.95 ms per loop
```

line_profiler in command line

The source code with @profile directive before the function we want to profile:

```
import requests

@profile
def slow_func():
    s = requests.session()
    html=s.get("https://en.wikipedia.org/").text
    sum([pow(ord(x),3.1) for x in list(html)])

for i in range(50):
    slow_func()
```

Using kernprof command to calculate profiling line by line

```
$ kernprof -lv so6.py

Wrote profile results to so6.py.lprof
Timer unit: 4.27654e-07 s

Total time: 22.6427 s
File: so6.py
Function: slow_func at line 4

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
     4                                           @profile
     5                                           def slow_func():
     6        50        20729    414.6      0.0      s = requests.session()
     7        50     47618627 952372.5     89.9      html=s.get("https://en.wikipedia.org/").tex
     8        50      5306958 106139.2     10.0      sum([pow(ord(x),3.1) for x in list(html)])
```

Page request is almost always slower than any calculation based on the information on the page.

timeit command line

Profiling concatanation of numbers

```
python -m timeit "'-'.join(str(n) for n in range(100))"
10000 loops, best of 3: 29.2 usec per loop
```

```
python -m timeit "'-'.join(map(str,range(100)))"
100000 loops, best of 3: 19.4 usec per loop
```

timeit() function

Profiling repetition of elements in an array

```
>>> import timeit
>>> timeit.timeit('list(itertools.repeat("a", 100))', 'import itertools', number = 10000000)
10.997665435877963
>>> timeit.timeit('["a"]*100', number = 10000000)
7.118789926862576
```

Using cProfile (Preferred Profiler)

Python includes a profiler called cProfile. This is generally preferred over using timeit.

It breaks down your entire script and for each method in your script it tells you:

- `ncalls` : The number of times a method was called
- `tottime` : Total time spent in the given function (excluding time made in calls to sub-functions)
- `percall` : Time spent per call. Or the quotient of tottime divided by ncalls
- `cumtime` : The cumulative time spent in this and all subfunctions (from invocation till exit). This figure is accurate even for recursive functions.
- `percall` : is the quotient of cumtime divided by primitive calls
- `filename:lineno(function)` : provides the respective data of each function

The cProfiler can be easily called on Command Line using:

```
$ python -m cProfile main.py
```

To sort the returned list of profiled methods by the time taken in the method:

```
$ python -m cProfile -s time main.py
```

Syntax

Parameters

Remarks