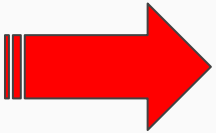
Easy and fast **Python Profiling**

Finding and solving performance bottlenecks

Table of contents

Finding **bottlenecks**

The needle in the haystack



Standard Python profiler

```
# Profiling

import cProfile
import re

cProfile.run('re.compile("foo|bar")',  # profiled with exec()
             'profiling_results')   # put it to a file
```

Standard Python profiler

```
# More control over profiling and no need for a results file

import cProfile
from custom_module import custom_function

profiler = cProfile.Profile()

profiler.runcall(custom_function, *args, **kwargs)

profiler.enable()
# ... do something ...
profiler.disable()
```

Formatting the profiling results

```
# Formatting result
import pstats

# Reading profiling data from file
p = pstats.Stats('profiling_results')
p.strip_dirs().sort_stats('time').print_stats()

# Reading profiling data from profiler
stats = pstats.Stats(profiler)
stats.strip_dirs().sort_stats('cumulative').print_stats()
```

Formatting the profiling results

```
Ordered by: cumulative time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.120    0.120    1.120    1.120 module.py:88(custom_func)
    41356    0.082    0.000    0.611    0.000 other.py:23(func)
    41356    0.089    0.000    0.475    0.000 other.py:59(other_method)
   213782    0.109    0.000    0.317    0.000 re.py:169(match)
    41356    0.224    0.000    0.232    0.000 other.py:330(_internal_y)
    41356    0.016    0.000    0.145    0.000 other.py:206(_internal_y)
   213782    0.112    0.000    0.112    0.000 re.py:286(_compile)
```

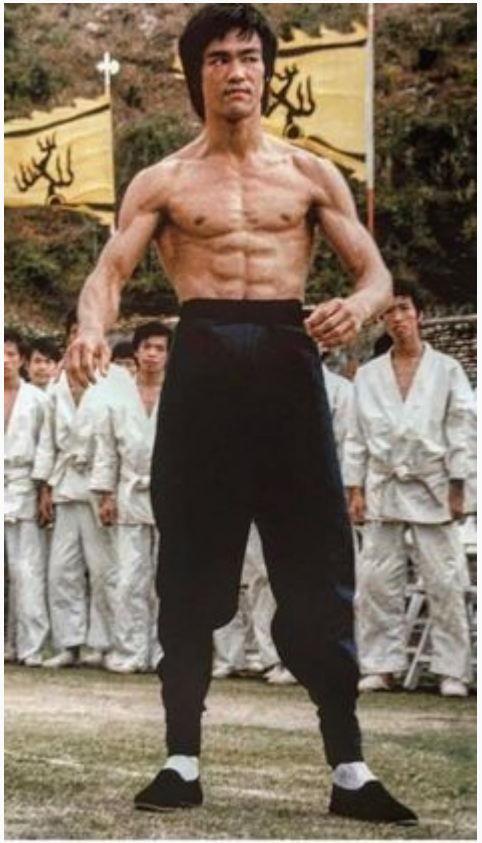
Finding **solutions**

Tiny and easy … but powerful

`timeit`

Tiny and easy … but powerful

`timeit`



Finding solutions with *timeit*

```
# The winner is ...

timeit ...something...

timeit (x for x in range(10000))
687 ns ± 15.2 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)

timeit [x for x in range(10000)]
341 µs ± 34.4 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

Finding solutions with *timeit*

```
# Important to know

1 sec (second)

1,000 ms (milliseconds)

1,000,000 µs (microseconds)

1,000,000,000 ns (nanoseconds)

1,000,000,000,000 ps (picoseconds)
```

Let’s code with Quiz time

Find and solve the performance issue

The example task:

- Parse a big log file
- Get time stamps with datetime objects
- Keep only the first occurance

See code snippets at the end.

Q&A

Fin

Examples

```
# --- utils.py ---

#! /usr/bin/env python
# -*- coding: utf-8 -*-

from cProfile import Profile
from pstats import Stats
from functools import wraps


def profiling(f):
    """Decorator for profiling"""
    @wraps(f)
    def decorated(*args, **kwargs):

        # Run profiling
        profiler = Profile()
        result = profiler.runcall(f, *args, **kwargs)

        stats = Stats(profiler)
        stats.strip_dirs()
        stats.sort_stats('cumulative')
        stats.print_stats(15)  # top x
        #stats.print_stats(.5)  # top 50 %

        return result

    return decorated
```

```
# --- profiling_x.py with versions ---

#! /usr/bin/env python
# -*- coding: utf-8 -*-

import re
from dateutil.parser import parse
from utils import profiling


@profiling

def profile_it():

    with open('big_logfile.log', 'r') as f:
        all_data = f.readlines()
        ct = 0

        entries = []

        for line in all_data:

            match = re.match(r'\[([-0-9\s:]+)\]', line)

            if match:
                dt_string = match.group(1)
            else:
                continue

            dt_obj = parse(dt_string)

            if dt_string not in [x for x, y in entries]:
                entries.append((dt_string, dt_obj))

            ct += 1

    return 'done!'


if __name__ == '__main__':
    profile_it()
```

```
# s = '2018-06-11 14:09:30'
# timeit parse(s)
# timeit datetime.strptime(s, '%Y-%m-%d %H:%M:%S')


@profiling
def profile_it():

    with open('big_logfile.log', 'r') as f:
        all_data = f.readlines()
        ct = 0

        entries = []

        for line in all_data:

            match = re.match(r'\[([-0-9\s:]+)\]', line)

            if match:
                dt_string = match.group(1)
            else:
                continue

            # dt_obj = parse(dt_string)
            dt_obj = datetime.strptime(dt_string, '%Y-%m-%d %H:%M:%S')
# TODO: NOTE THIS *******
            if dt_string not in [x for x, y in entries]:
                entries.append((dt_string, dt_obj))

            ct += 1

    return 'done!'
```

Examples

```
@profiling
def profile_it():

    with open('big_logfile.log', 'r') as f:
        all_data = f.readlines()
        ct = 0

        entries = []
        used = []

        for line in all_data:

            match = re.match(r'\[([-0-9\s:]+)\]', line)

            if match:
                dt_string = match.group(1)
            else:
                continue

            dt_obj = datetime.strptime(dt_string, '%Y-%m-%d %H:%M:%S')

            # if dt_string not in [x for x, y in entries]:
            if dt_string not in used:    # TODO: NOTE THIS *******
                entries.append((dt_string, dt_obj))
                used.append(dt_string)

            ct += 1

    return 'done!'
```

```
@profiling
def profile_it():

    with open('big_logfile.log', 'r') as f:
        all_data = f.readlines()
        ct = 0

        entries = {}

        for line in all_data:

            match = re.match(r'\[([-0-9\s:]+)\]', line)

            if match:
                dt_string = match.group(1)
            else:
                continue

            dt_obj = datetime.strptime(dt_string, '%Y-%m-%d %H:%M:%S')

            # if dt_string not in used:
            if dt_string not in entries.keys():  # TODO: NOTE THIS
*******
                entries[dt_string] = dt_obj

            ct += 1

    return 'done!'
```

```
# s = '2018-06-11 14:09:30'
# timeit datetime.strptime(s, '%Y-%m-%d %H:%M:%S')
# timeit isoparse(s)


@profiling
def profile_it():

    with open('big_logfile.log', 'r') as f:
        all_data = f.readlines()
        ct = 0

        entries = {}

        for line in all_data:

            match = re.match(r'\[([-0-9\s:]+)\]', line)

            if match:
                dt_string = match.group(1)
            else:
                continue

            # dt_obj = datetime.strptime(dt_string, '%Y-%m-%d
%H:%M:%S')
            dt_obj = isoparse(dt_string)  # TODO: NOTE THIS *******

            if dt_string not in entries.keys():
                entries[dt_string] = dt_obj

            ct += 1

    return 'done!'
```

Examples

```
# s = '2018-06-11 14:09:30'
# timeit isoparse(s)
# timeit parse_datetime(s)


@profiling
def profile_it():

    with open('big_logfile.log', 'r') as f:
        all_data = f.readlines()
        ct = 0

        entries = {}

        for line in all_data:

            match = re.match(r'\[([-0-9\s:]+)\]', line)

            if match:
                dt_string = match.group(1)
            else:
                continue

            # dt_obj = isoparse(dt_string)
            dt_obj = parse_datetime(dt_string)  # TODO: NOTE THIS *******

            if dt_string not in entries.keys():
                entries[dt_string] = dt_obj

            ct += 1

    return 'done!'
```

```
# s = '[2018-06-11 14:09:30] ERROR [wpm.wpm.middleware:process_request:212]
Failed to create APIRequestLog.'
# timeit re.match(r'\[([-0-9\s:]+)\]', s)
# pattern = re.compile(r'\[([-0-9\s:]+)\]')
# timeit pattern.match(s)


@profiling
def profile_it():

    with open('big_logfile.log', 'r') as f:
        all_data = f.readlines()
        ct = 0

        entries = {}
        pattern = re.compile(r'\[([-0-9\s:]+)\]')
        #pattern = re.compile(r'\[(\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2})\]')

        for line in all_data:

            # match = re.match(r'\[([-0-9\s:]+)\]', line)
            match = pattern.match(line)  # TODO: NOTE THIS *******

            if match:
                dt_string = match.group(1)
            else:
                continue

            dt_obj = parse_datetime(dt_string)

            if dt_string not in entries.keys():
                entries[dt_string] = dt_obj

            ct += 1

    return 'done!'
```