Examining runtime

Why should we time our code?

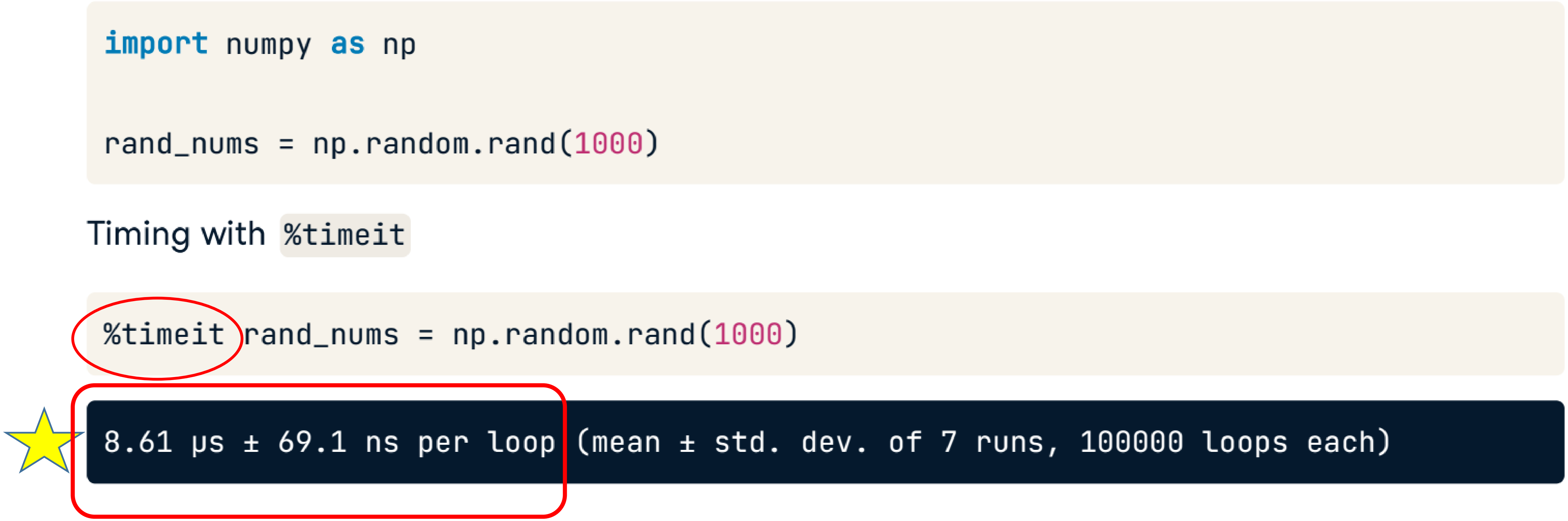
- Allows us to pick the optimal coding approach
- Faster code == more effcient code!

How can we time our code?

- Calculate runtime with IPython magic command %timeit
- Magic commands: enhancements on top of normal Python syntax
  - ✓ Prefixed by the "%" character
  - ✓ Link to docs (here)
  - ✓ See all available magic commands with %lsmagic

Using %timeit

- Code to be timed



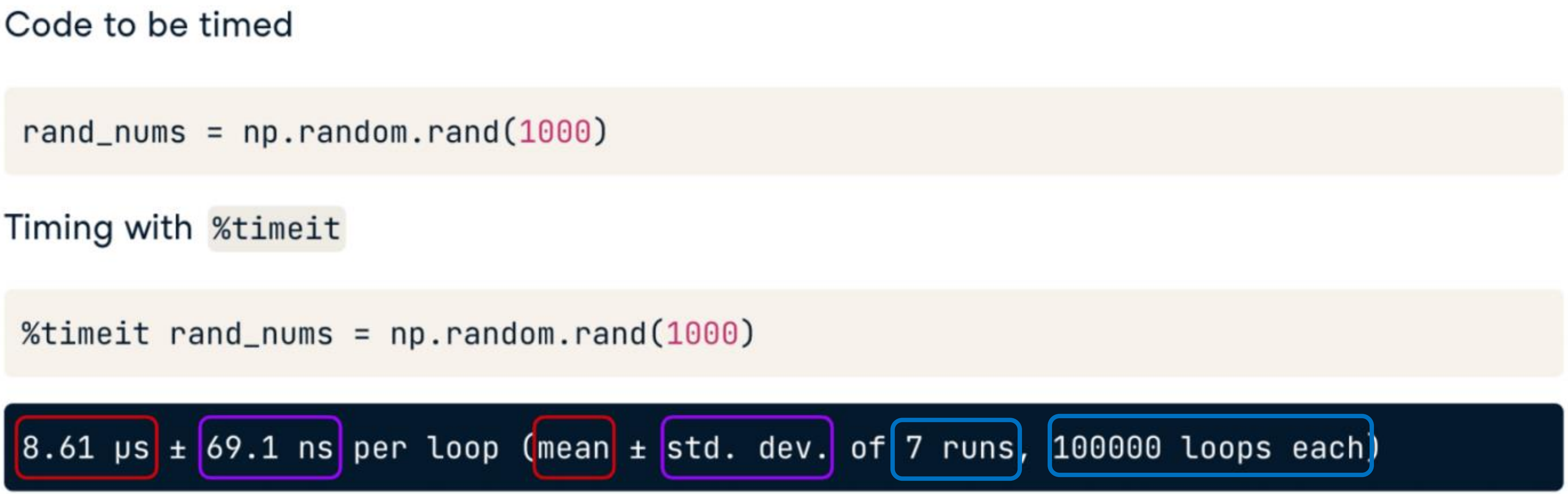
```
import numpy as np

rand_nums = np.random.rand(1000)
```

Timing with `%timeit`

```
%timeit rand_nums = np.random.rand(1000)
```

```
8.61 µs ± 69.1 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

%timeit output



Code to be timed

```
rand_nums = np.random.rand(1000)
```

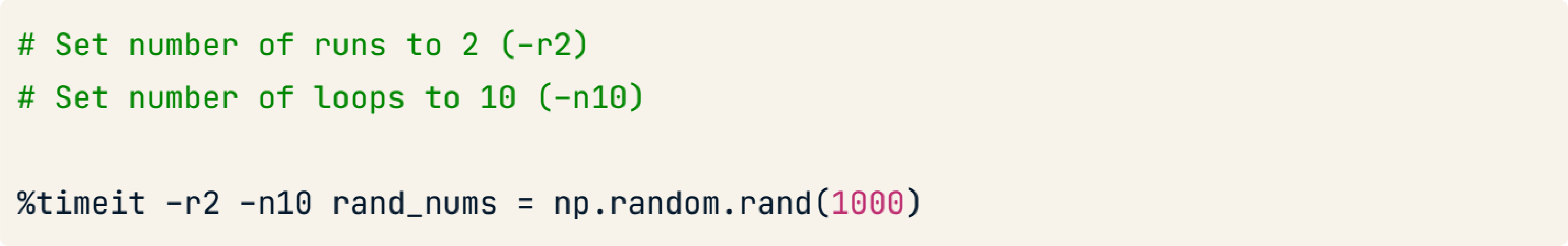
Timing with `%timeit`

```
%timeit rand_nums = np.random.rand(1000)
```

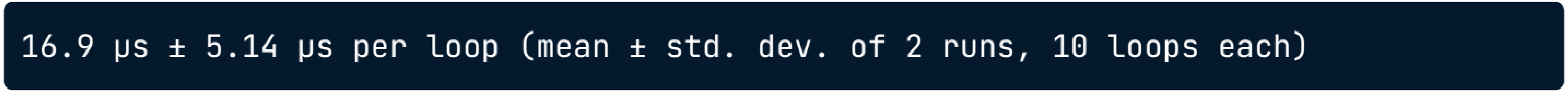
```
8.61 µs ± 69.1 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

Specifying number of runs/loops

- Setting the number of runs (-r) and/or loops (-n)



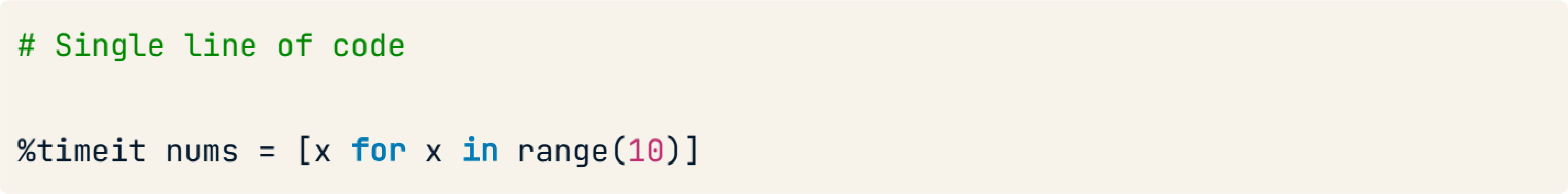
```
# Set number of runs to 2 (-r2)
# Set number of loops to 10 (-n10)

%timeit -r2 -n10 rand_nums = np.random.rand(1000)
```



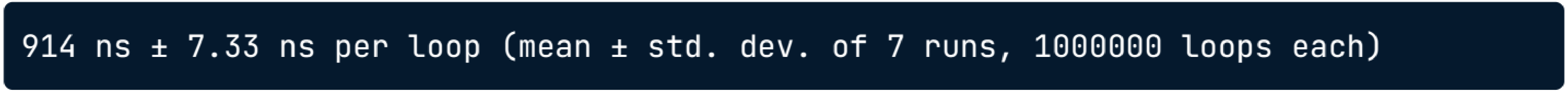
```
16.9 µs ± 5.14 µs per loop (mean ± std. dev. of 2 runs, 10 loops each)
```

Using %timeit in line magic mode

- Line magic (%timeit)



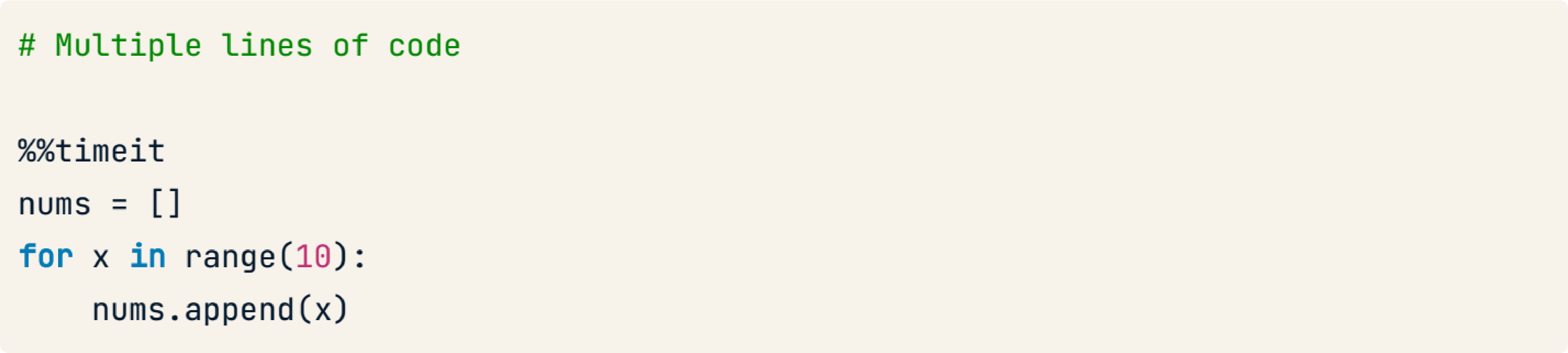
```
# Single line of code

%timeit nums = [x for x in range(10)]
```



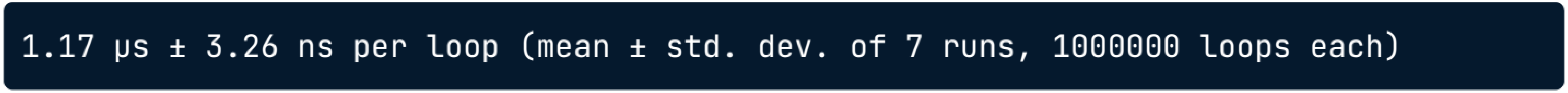
```
914 ns ± 7.33 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)
```

Using %timeit in cell magic mode

- Cell magic (%%timeit)



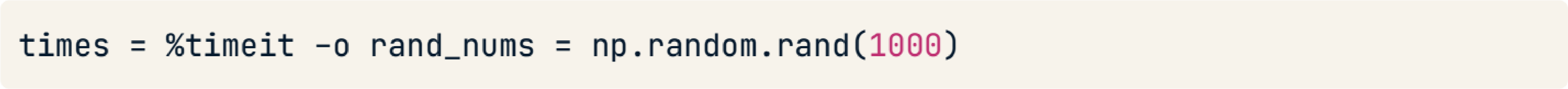
```
# Multiple lines of code

%%timeit
nums = []
for x in range(10):
    nums.append(x)
```



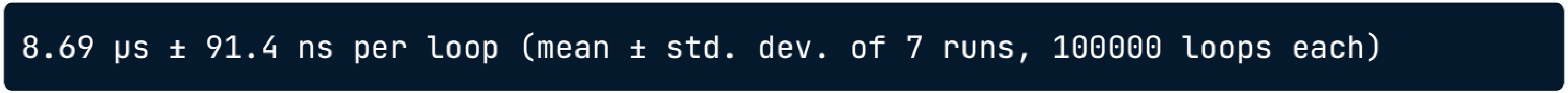
```
1.17 µs ± 3.26 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)
```

Saving output

- Saving the output to a variable (-o)

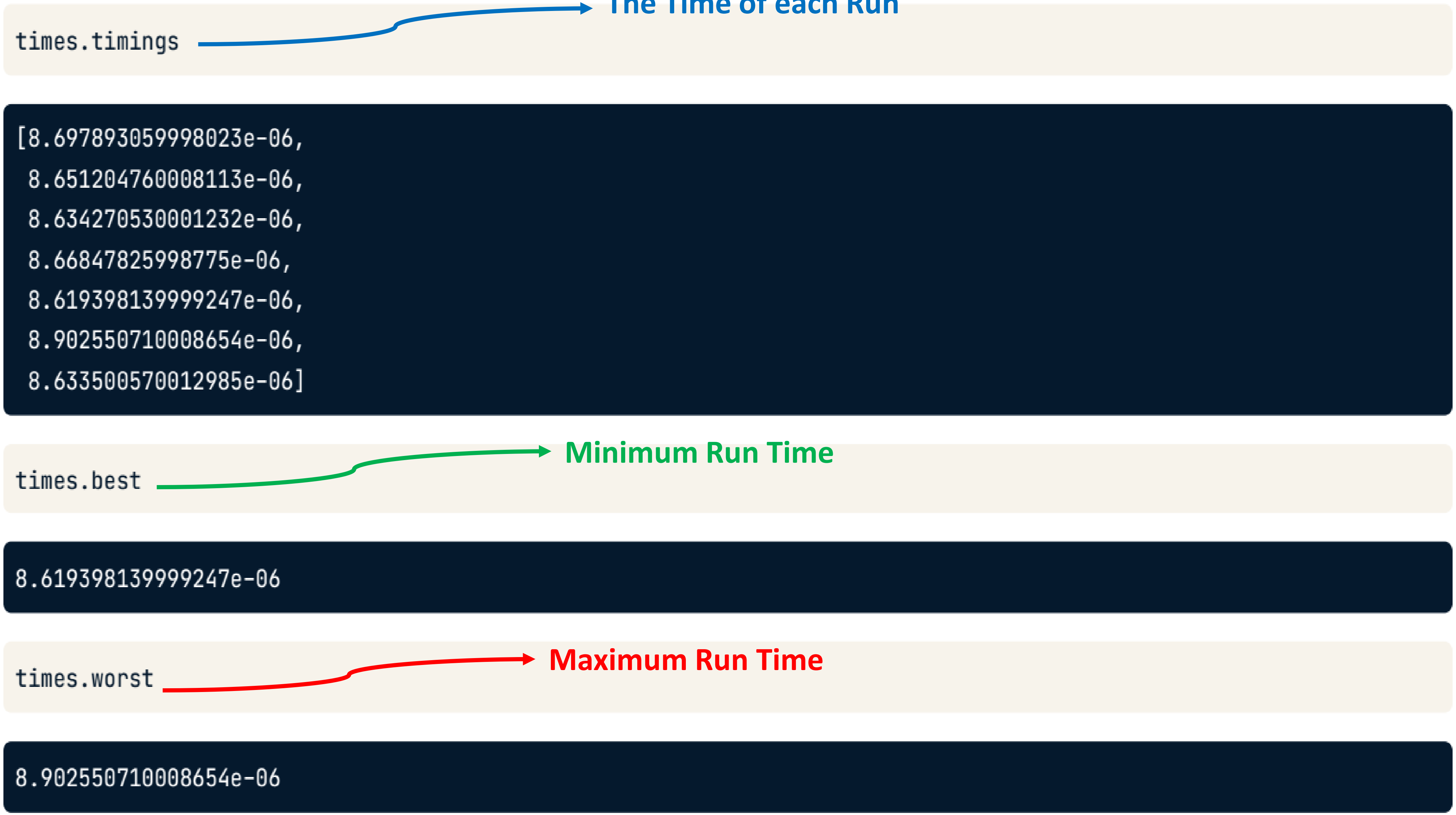


```
times = %timeit -o rand_nums = np.random.rand(1000)
```



```
8.69 μs ± 91.4 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

Saving output



```
times.timings
```

The Time of each Run

```
[8.697893059998023e-06,
 8.651204760008113e-06,
 8.634270530001232e-06,
 8.66847825998775e-06,
 8.619398139999247e-06,
 8.902550710008654e-06,
 8.633500570012985e-06]
```

```
times.best
```

Minimum Run Time

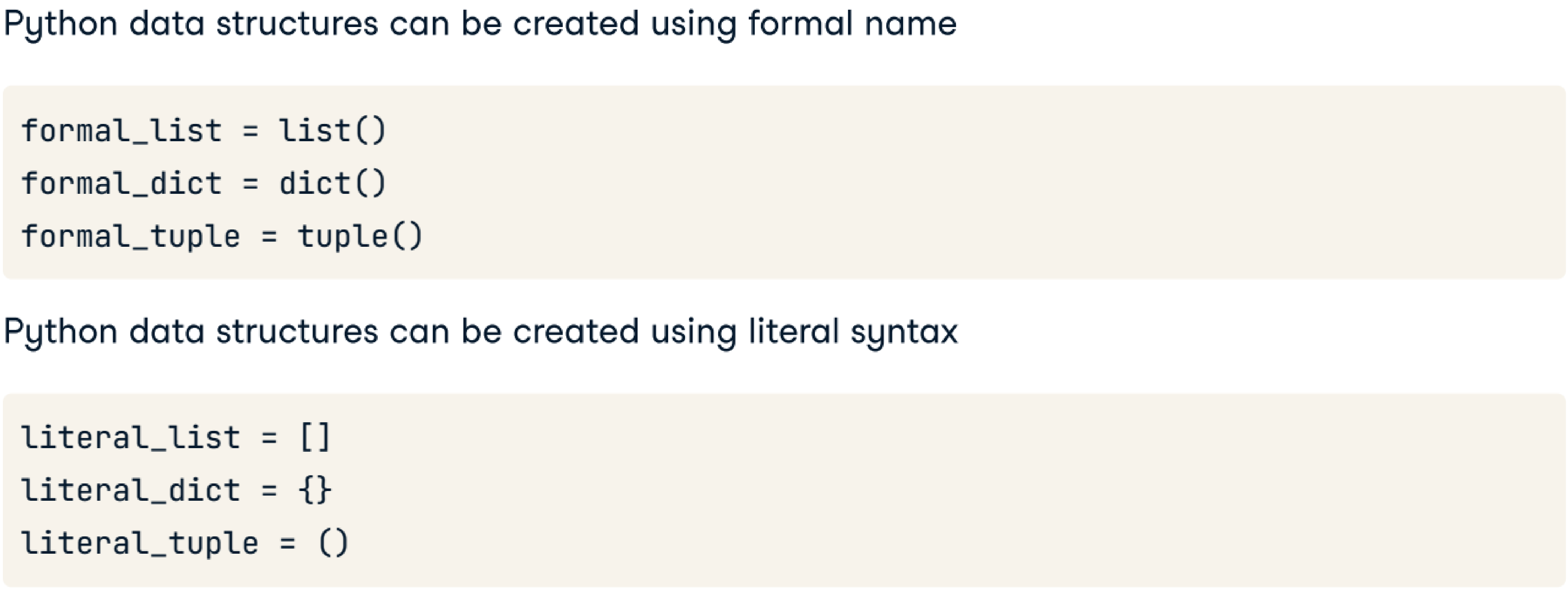
```
8.619398139999247e-06
```

```
times.worst
```

Maximum Run Time

```
8.902550710008654e-06
```

Comparing times



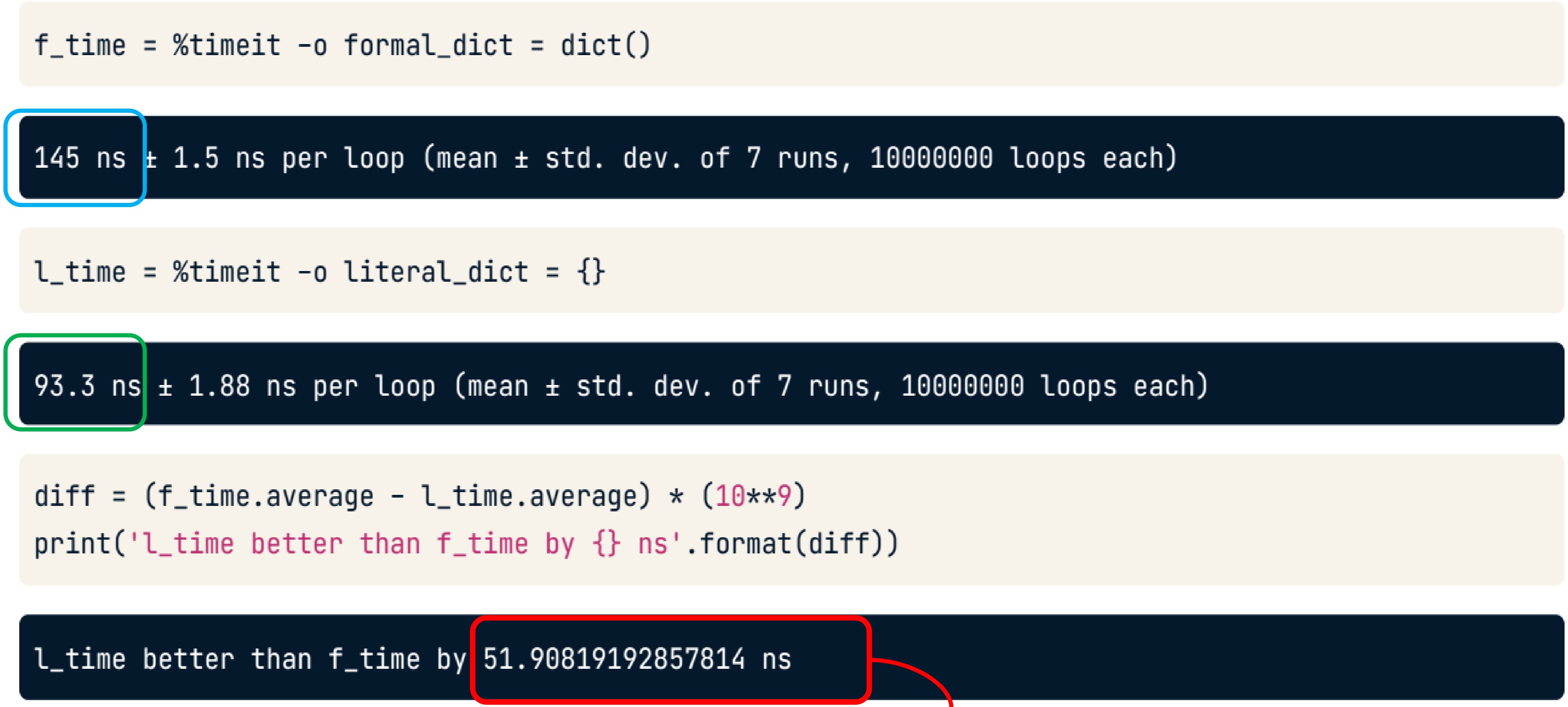
Python data structures can be created using formal name

```
formal_list = list()
formal_dict = dict()
formal_tuple = tuple()
```

Python data structures can be created using literal syntax

```
literal_list = []
literal_dict = {}
literal_tuple = ()
```

Comparing times



```
f_time = %timeit -o formal_dict = dict()
```

```
145 ns ± 1.5 ns per loop (mean ± std. dev. of 7 runs, 10000000 loops each)
```

```
l_time = %timeit -o literal_dict = {}
```

```
93.3 ns ± 1.88 ns per loop (mean ± std. dev. of 7 runs, 10000000 loops each)
```

```
diff = (f_time.average - l_time.average) * (10**9)
print('l_time better than f_time by {} ns'.format(diff))
```

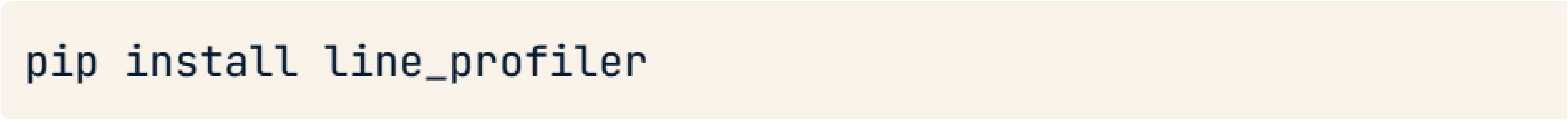
```
l_time better than f_time by 51.90819192857814 ns
```

Literal is Faster than Formal

Code profiling for runtime

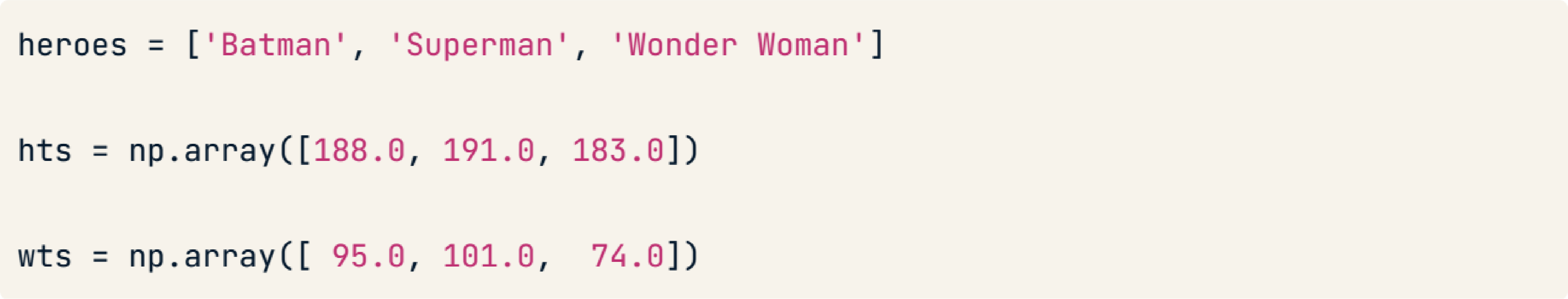
Code profiling

- Detailed stats on frequency and duration of function calls
- Line-by-line analyses
- Package used: line_profiler



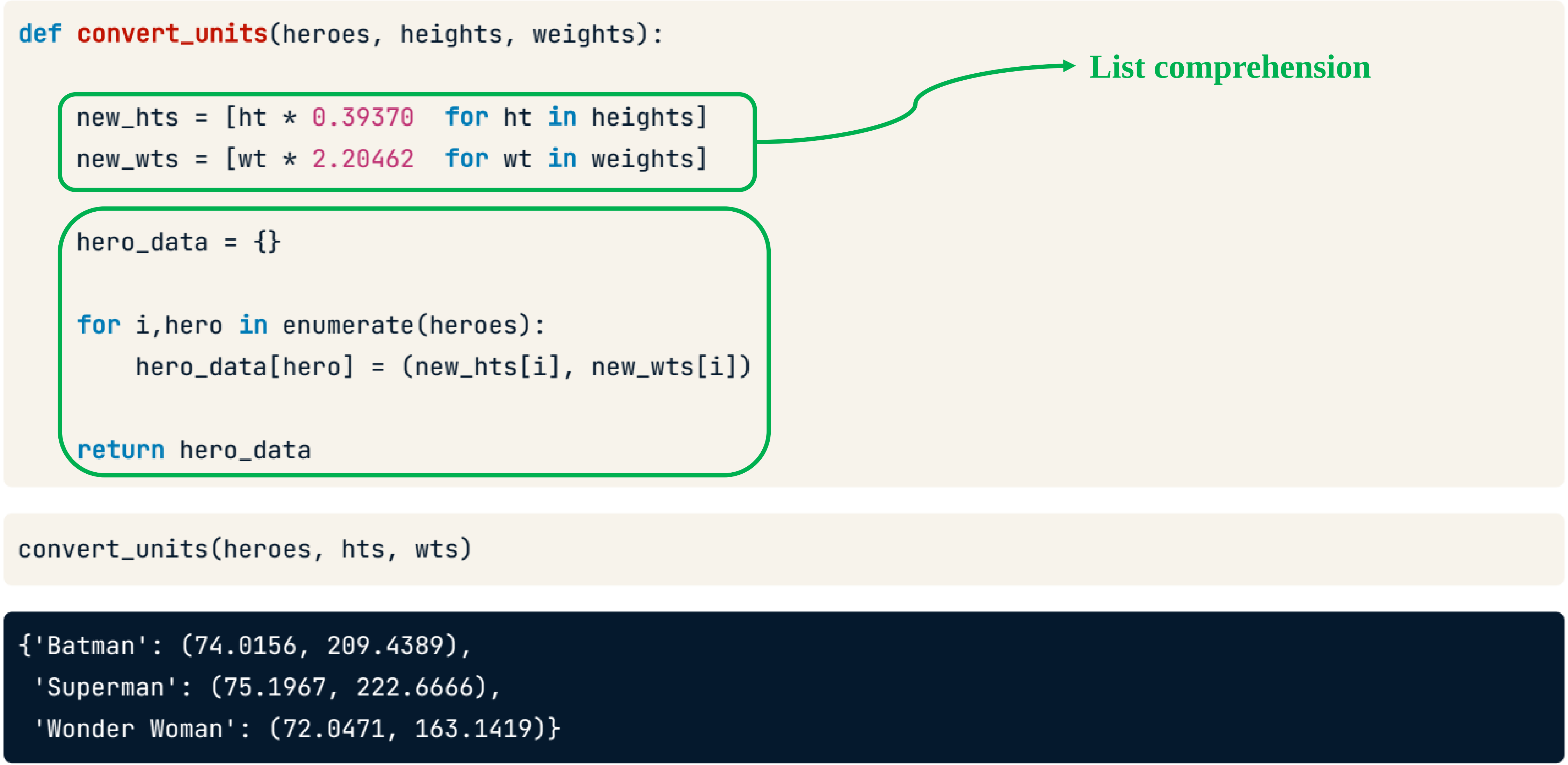
```
pip install line_profiler
```

Code profiling: runtime



```
heroes = ['Batman', 'Superman', 'Wonder Woman']

hts = np.array([188.0, 191.0, 183.0])

wts = np.array([ 95.0, 101.0,  74.0])
```

Code profiling: runtime



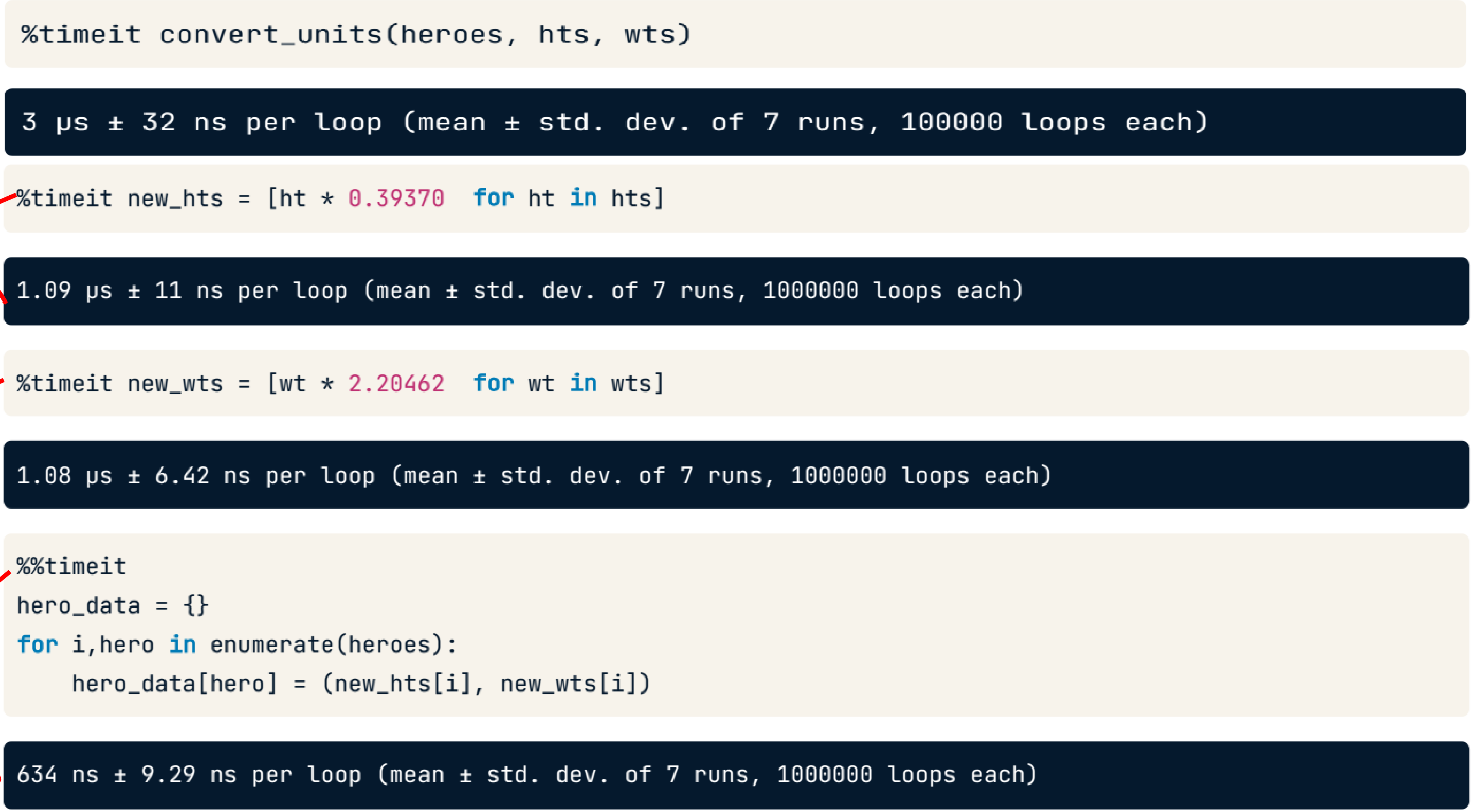
```
def convert_units(heroes, heights, weights):

    new_hts = [ht * 0.39370  for ht in heights]
    new_wts = [wt * 2.20462  for wt in weights]

    hero_data = {}

    for i,hero in enumerate(heroes):
        hero_data[hero] = (new_hts[i], new_wts[i])

    return hero_data
```

List comprehension

```
convert_units(heroes, hts, wts)
```

```
{'Batman': (74.0156, 209.4389),
 'Superman': (75.1967, 222.6666),
 'Wonder Woman': (72.0471, 163.1419)}
```

Code profiling: runtime



```
%timeit convert_units(heroes, hts, wts)
```

```
3 µs ± 32 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

```
%timeit new_hts = [ht * 0.39370  for ht in hts]
```

```
1.09 µs ± 11 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)
```

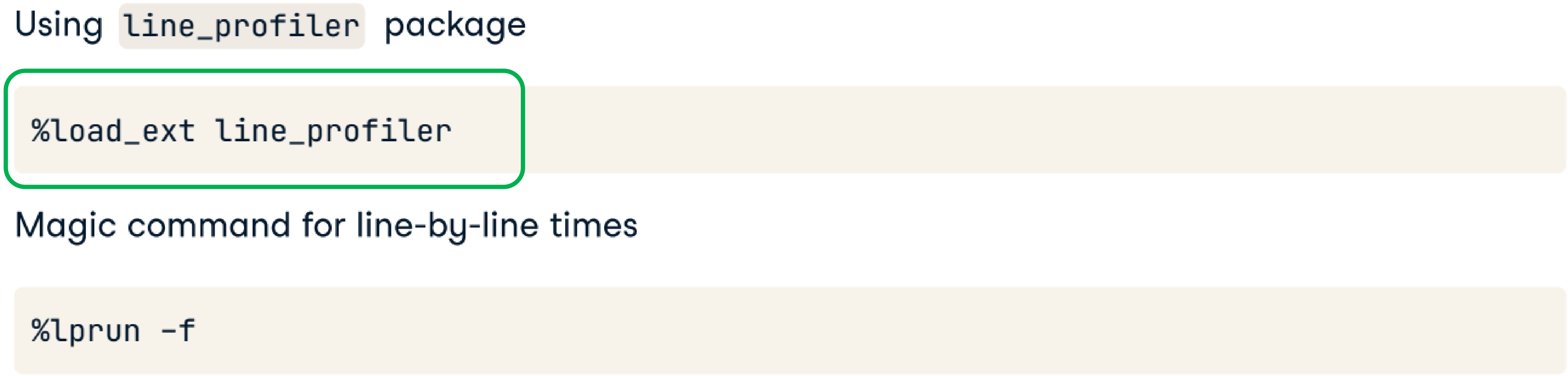
```
%timeit new_wts = [wt * 2.20462  for wt in wts]
```

```
1.08 µs ± 6.42 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)
```

```
%%timeit
hero_data = {}
for i,hero in enumerate(heroes):
    hero_data[hero] = (new_hts[i], new_wts[i])
```

```
634 ns ± 9.29 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)
```

Code profiling: line_profiler



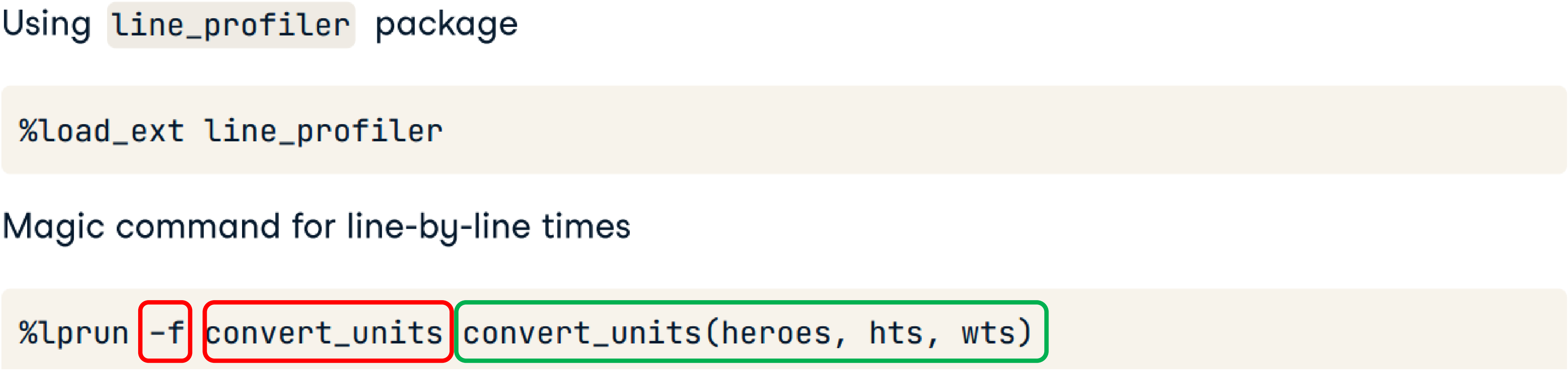
Using `line_profiler` package

```
%load_ext line_profiler
```

Magic command for line-by-line times

```
%lprun -f
```

Code profiling: line_profiler



Using `line_profiler` package

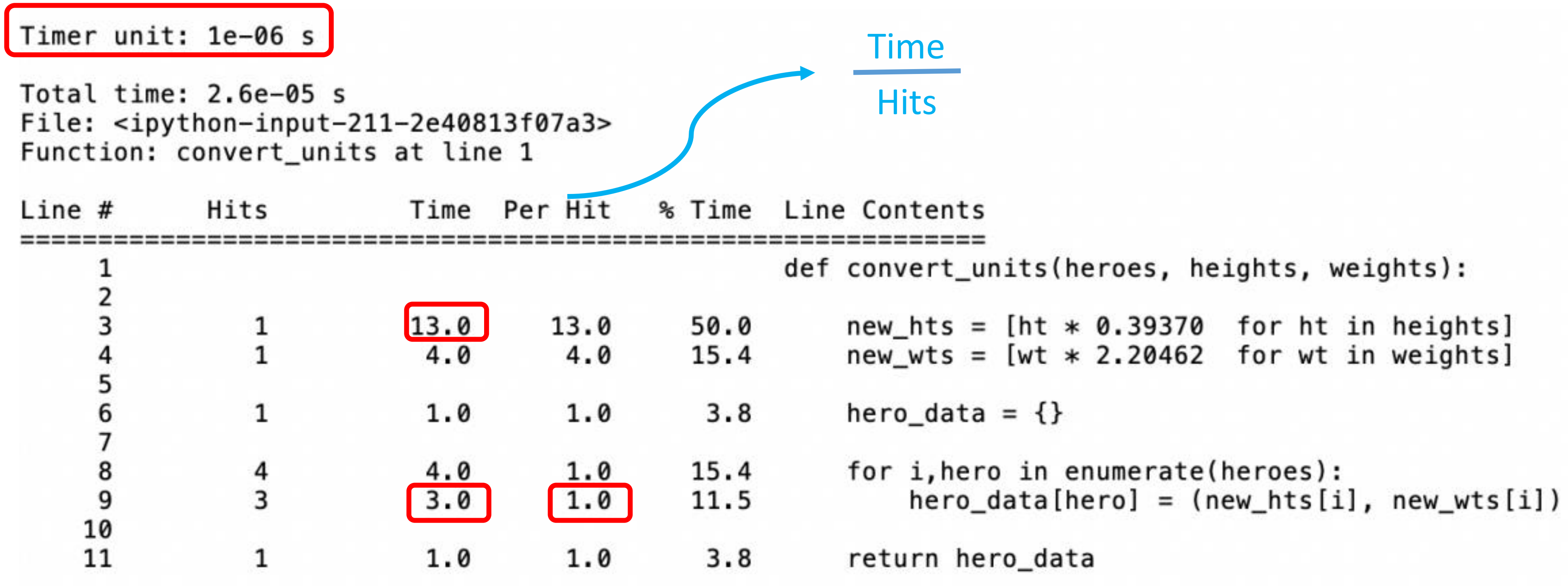
```
%load_ext line_profiler
```

Magic command for line-by-line times

```
%lprun -f convert_units convert_units(heroes, hts, wts)
```

%lprun output

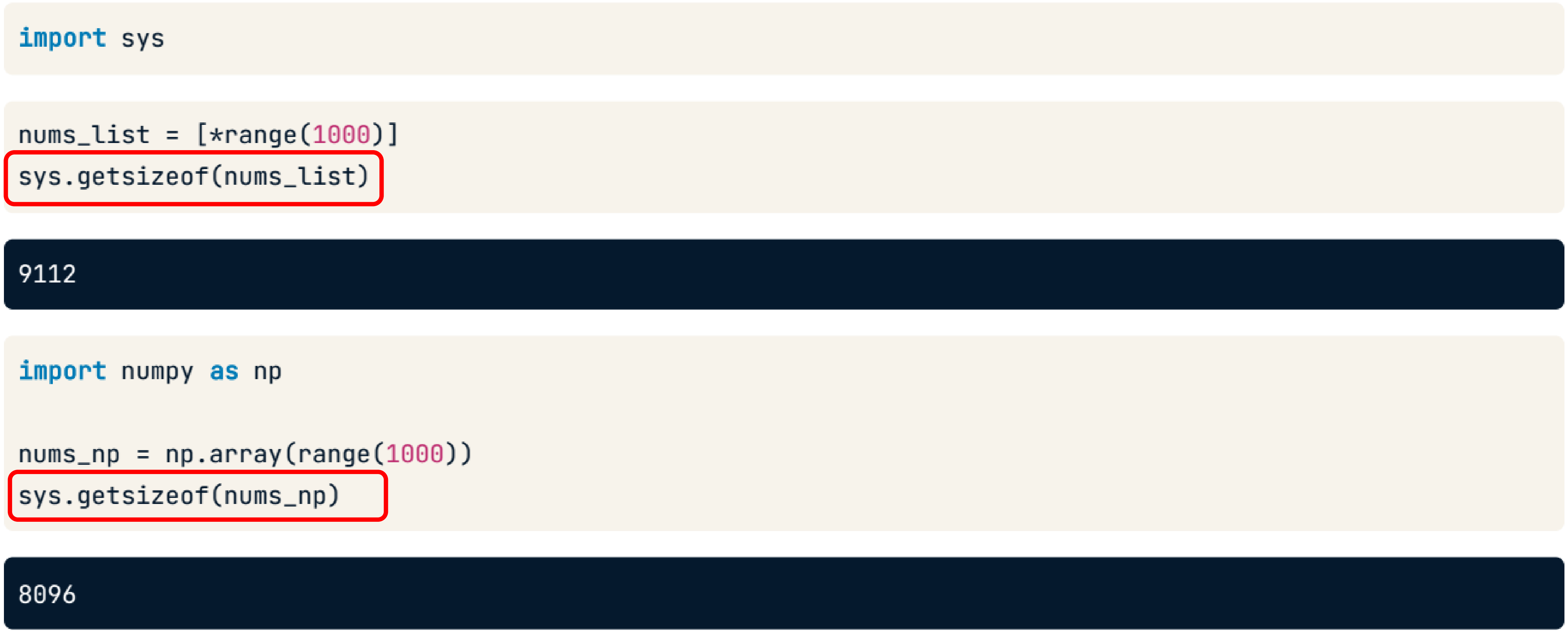
```
%lprun -f convert_units convert_units(heroes, hts, wts)
```



```
Timer unit: 1e-06 s

Total time: 2.6e-05 s
File: <ipython-input-211-2e40813f07a3>
Function: convert_units at line 1

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
     1                                           def convert_units(heroes, heights, weights):
     2
     3         1         13.0     13.0     50.0      new_hts = [ht * 0.39370  for ht in heights]
     4         1          4.0      4.0     15.4      new_wts = [wt * 2.20462  for wt in weights]
     5
     6         1          1.0      1.0      3.8      hero_data = {}
     7
     8         4          4.0      1.0     15.4      for i,hero in enumerate(heroes):
     9         3          3.0      1.0     11.5          hero_data[hero] = (new_hts[i], new_wts[i])
    10
    11         1          1.0      1.0      3.8      return hero_data
```

Code profiling for memory usage

Quick and dirty approach



```
import sys
```

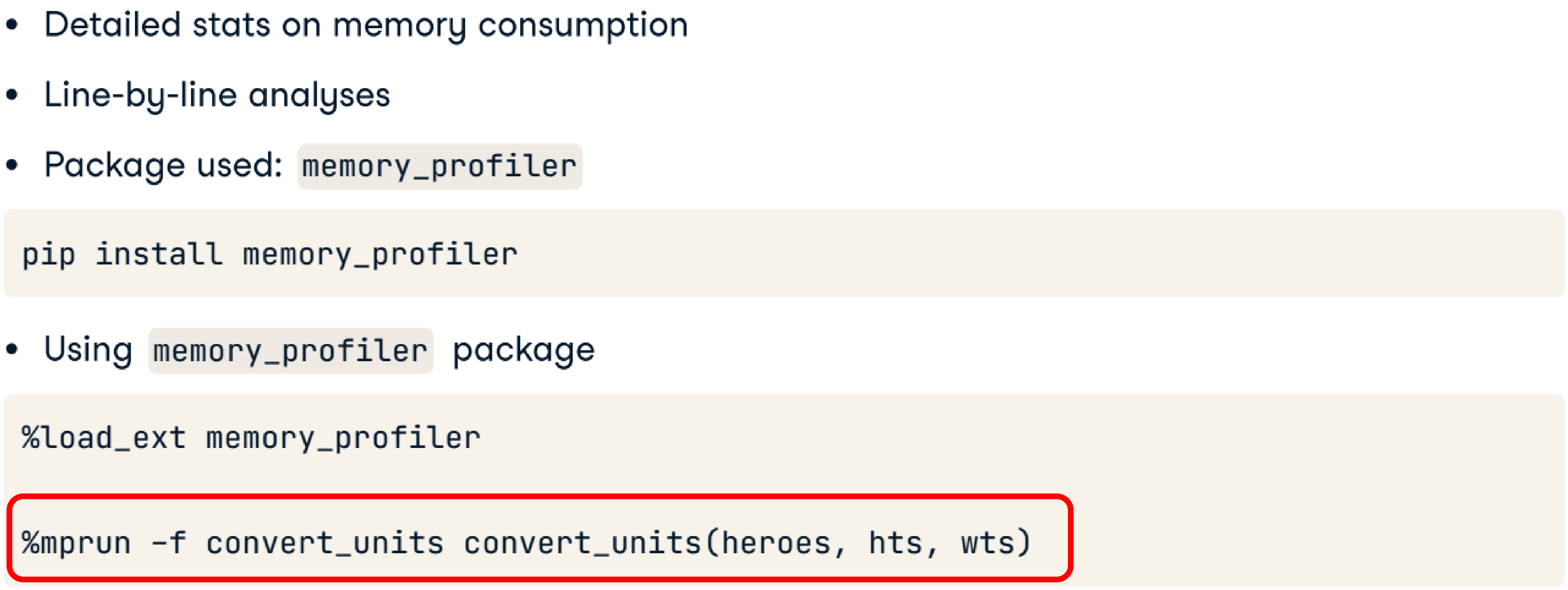
```
nums_list = [*range(1000)]
sys.getsizeof(nums_list)
```

```
9112
```

```
import numpy as np

nums_np = np.array(range(1000))
sys.getsizeof(nums_np)
```

```
8096
```

Code profiling: memory



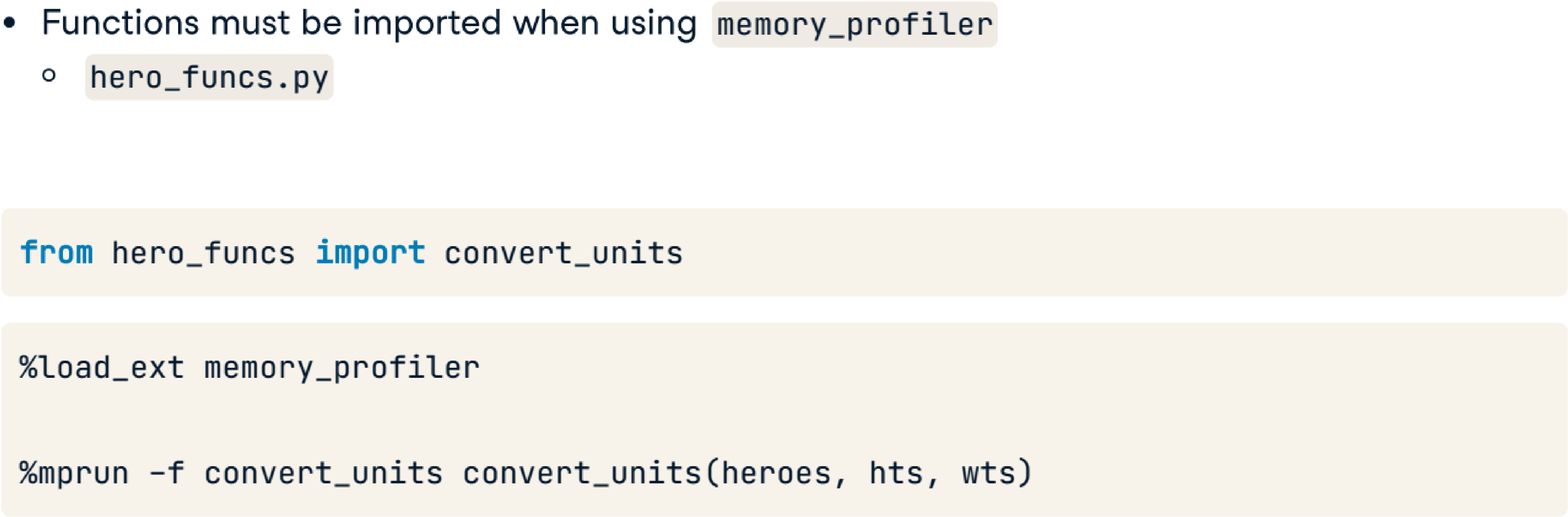
- Detailed stats on memory consumption
- Line-by-line analyses
- Package used: `memory_profiler`

```
pip install memory_profiler
```

- Using `memory_profiler` package

```
%load_ext memory_profiler

%mprun -f convert_units convert_units(heroes, hts, wts)
```

Code profiling: memory

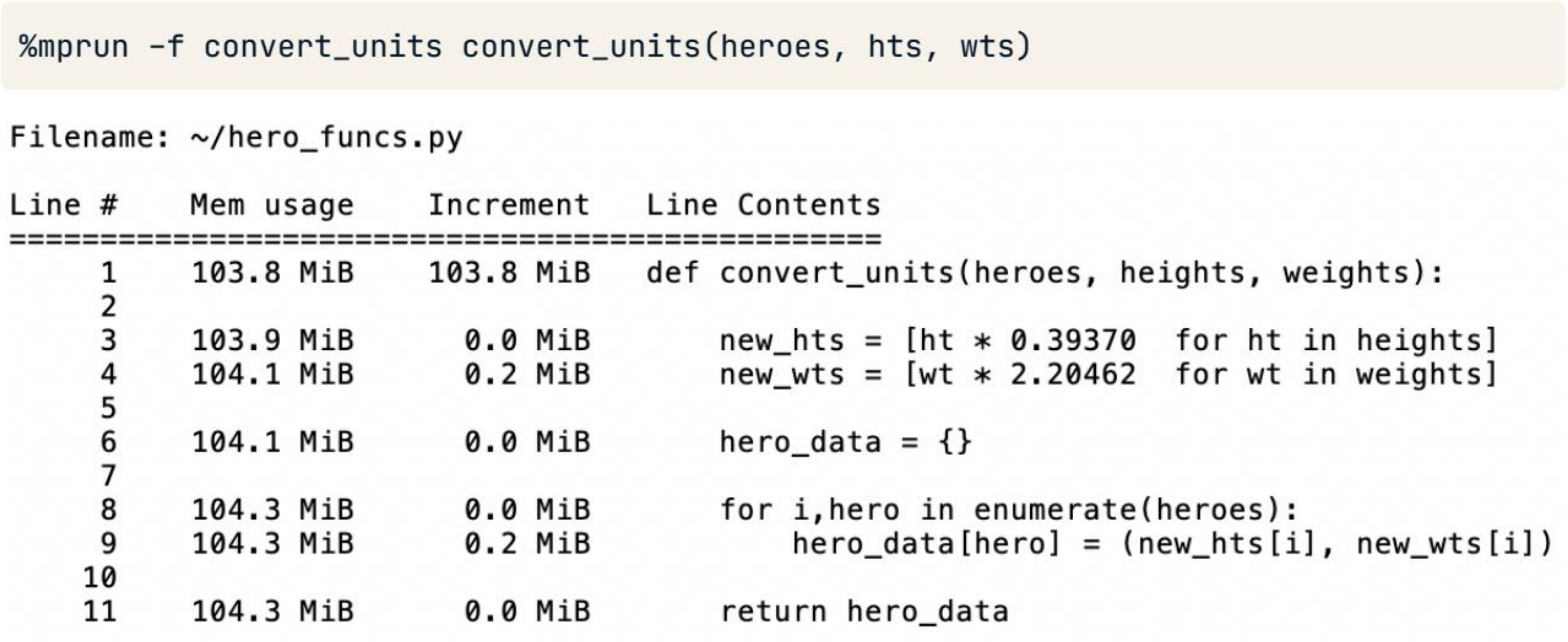


- Functions must be imported when using `memory_profiler`
  - `hero_funcs.py`

```
from hero_funcs import convert_units
```

```
%load_ext memory_profiler

%mprun -f convert_units convert_units(heroes, hts, wts)
```

%mprun output

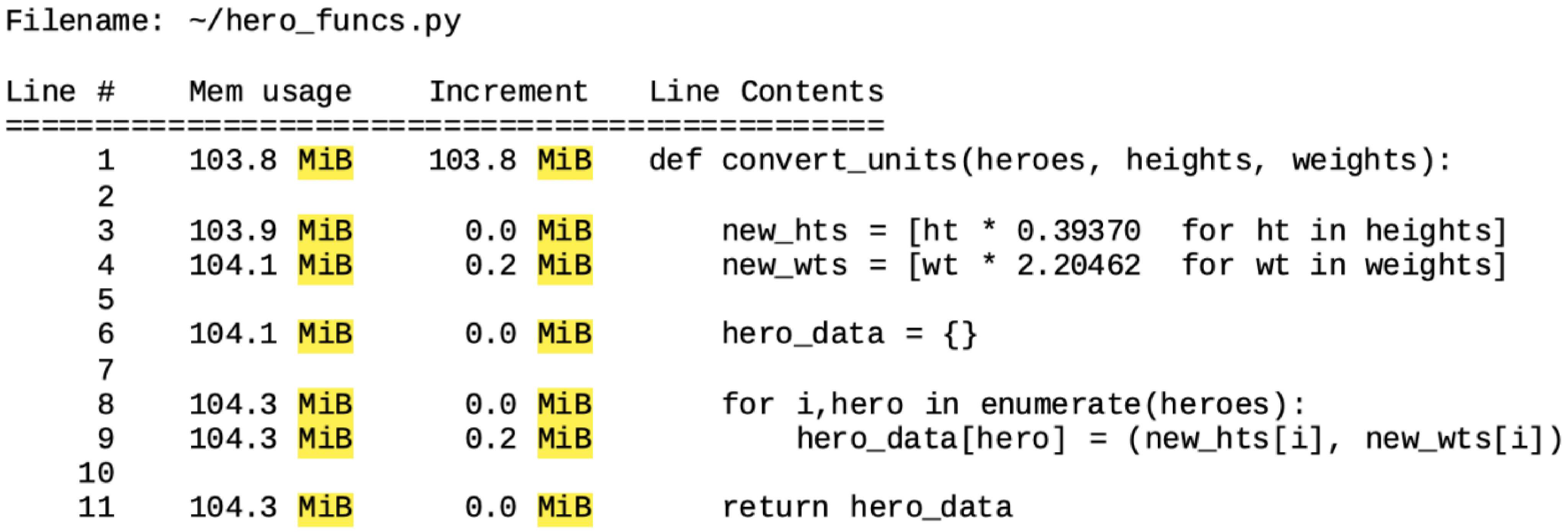


```
%mprun -f convert_units convert_units(heroes, hts, wts)
```

```
Filename: ~/hero_funcs.py

Line #    Mem usage    Increment   Line Contents
================================================
     1    103.8 MiB    103.8 MiB   def convert_units(heroes, heights, weights):
     2
     3    103.9 MiB      0.0 MiB       new_hts = [ht * 0.39370  for ht in heights]
     4    104.1 MiB      0.2 MiB       new_wts = [wt * 2.20462  for wt in weights]
     5
     6    104.1 MiB      0.0 MiB       hero_data = {}
     7
     8    104.3 MiB      0.0 MiB       for i,hero in enumerate(heroes):
     9    104.3 MiB      0.2 MiB           hero_data[hero] = (new_hts[i], new_wts[i])
    10
    11    104.3 MiB      0.0 MiB       return hero_data
```

%mprun output caveats

```
%mprun -f convert_units convert_units(heroes, hts, wts)
```



```
Filename: ~/hero_funcs.py

Line #    Mem usage    Increment   Line Contents
================================================
     1    103.8 MiB    103.8 MiB   def convert_units(heroes, heights, weights):
     2
     3    103.9 MiB      0.0 MiB       new_hts = [ht * 0.39370  for ht in heights]
     4    104.1 MiB      0.2 MiB       new_wts = [wt * 2.20462  for wt in weights]
     5
     6    104.1 MiB      0.0 MiB       hero_data = {}
     7
     8    104.3 MiB      0.0 MiB       for i,hero in enumerate(heroes):
     9    104.3 MiB      0.2 MiB           hero_data[hero] = (new_hts[i], new_wts[i])
    10
    11    104.3 MiB      0.0 MiB       return hero_data
```

Let's Continue!

Microsoft Bing