OUTLINE

- About Python

OUTLINE

- About Python
- Python is slow !

OUTLINE

- About Python
- Python is slow !
- Profiling a Python code

ABOUT PYTHON

- Python was created by Guido van Rossum in 1991 (last version 3.11 - 24/10/2022)

ABOUT PYTHON

- Python was created by Guido van Rossum in 1991 (last version 3.11 - 24/10/2022)
- Python is simple

ABOUT PYTHON

- Python was created by Guido van Rossum in 1991 (last version 3.11 - 24/10/2022)
- Python is simple
- Python is fully featured

ABOUT PYTHON

- Python was created by Guido van Rossum in 1991 (last version 3.11 - 24/10/2022)
- Python is simple
- Python is fully featured
- Python is readable

ABOUT PYTHON

- Python was created by Guido van Rossum in 1991 (last version 3.11 - 24/10/2022)
- Python is simple
- Python is fully featured
- Python is readable
- Python is extensible

ABOUT PYTHON

- Python was created by Guido van Rossum in 1991 (last version 3.11 - 24/10/2022)
- Python is simple
- Python is fully featured
- Python is readable
- Python is extensible
- Python is ubiquitous, portable, and free

ABOUT PYTHON

- Python was created by Guido van Rossum in 1991 (last version 3.11 - 24/10/2022)
- Python is simple
- Python is fully featured
- Python is readable
- Python is extensible
- Python is ubiquitous, portable, and free
- Python has many third party libraries, tools, and a large community

ABOUT PYTHON

- Python was created by Guido van Rossum in 1991 (last version 3.11 - 24/10/2022)
- Python is simple
- Python is fully featured
- Python is readable
- Python is extensible
- Python is ubiquitous, portable, and free
- Python has many third party libraries, tools, and a large community

➠ But Python is slow!!

ABOUT PYTHON

- Python was created by Guido van Rossum in 1991 (last version 3.11 - 24/10/2022)
- Python is simple
- Python is fully featured
- Python is readable
- Python is extensible
- Python is ubiquitous, portable, and free
- Python has many third party libraries, tools, and a large community

➠ When does it really matter?

PYTHON IS SLOW

When does it matter?

- Is my code fast?

PYTHON IS SLOW

When does it matter?

- Is my code fast?
- How many CPUh?

PYTHON IS SLOW

When does it matter?

- Is my code fast?
- How many CPUh?
- Problems on the system?

PYTHON IS SLOW

When does it matter?

- Is my code fast?
- How many CPUh?
- Problems on the system?
- How much effort is it to make it run faster?

PROFILING A PYTHON CODE: WHY?

- Code bottlenecks

PROFILING A PYTHON CODE: WHY?

- Code bottlenecks
- Premature optimization is the root of all evil D. Knuth

PROFILING A PYTHON CODE: WHY?

- Code bottlenecks
- Premature optimization is the root of all evil D. Knuth
- First make it work. Then make it right. Then make it fast. K. Beck

PROFILING A PYTHON CODE: WHY?

- Code bottlenecks
- Premature optimization is the root of all evil D. Knuth
- First make it work. Then make it right. Then make it fast. K. Beck
- How?

PROFILING A PYTHON CODE: PROFILERS

- Deterministic and statistical profiling

PROFILING A PYTHON CODE: PROFILERS

- Deterministic and statistical profiling
  - the profiler will be monitoring all the events

PROFILING A PYTHON CODE: PROFILERS

- Deterministic and statistical profiling
  - the profiler will be monitoring all the events
  - it will sample after time intervals to collect that information

PROFILING A PYTHON CODE: PROFILERS

- Deterministic and statistical profiling
  - the profiler will be monitoring all the events
  - it will sample after time intervals to collect that information
- The level at which resources are measured; module, function or line level

PROFILING A PYTHON CODE: PROFILERS

- Deterministic and statistical profiling
  - the profiler will be monitoring all the events
  - it will sample after time intervals to collect that information
- The level at which resources are measured; module, function or line level
- Profile viewers

PROFILING A PYTHON CODE: TOOLS

- Inbuilt timing modules
- profile and cProfile
- pstats
- line_profiler
- snakeviz

PROFILING A PYTHON CODE: USE CASE

```
def linspace(start, stop, n):
  step =float(stop -start) / (n -1)
  return [start +i *step for i in range(n)]

def mandel(c, maxiter):
  z =c
  for n in range(maxiter):
    if abs(z) >2:
      return n
    z =z*z +c
  return n

def mandel_set(xmin=-2.0, xmax=0.5, ymin=-1.25, ymax=1.25,
               width=1000, height=1000, maxiter=80):
  r =linspace(xmin, xmax, width)
  i =linspace(ymin, ymax, height)
  n =[[0]*width for _ in range(height)]
  for x in range(width):
    for y in range(height):
      n[y][x] =mandel(complex(r[x], i[y]), maxiter)
  return n
```

PROFILING A PYTHON CODE: TIMEIT

The very naive way

```
import timeit

start_time =timeit.default_timer()
mandel_set()
end_time =timeit.default_timer()
# Time taken in seconds
elapsed_time =end_time -start_time

print('> Elapsed time', elapsed_time)
```

or using the magic method `timeit`

```
[In] %timeit mandel_set()
[Out] 3.01 s +/- 84.6 ms per loop (mean +/- std. dev. of 7 runs, 1 loop each)
```

PROFILING A PYTHON CODE: PRUN

```
[In] %prun -s cumulative mandel_set()
```

which is, in console mode, equivalent to

```
python -m cProfile -s cumulative mandel.py
```

```
         25214601 function calls in 5.151 seconds

   Ordered by: cumulative time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    5.151    5.151 {built-in method builtins.exec}
        1    0.002    0.002    5.151    5.151 <string>:1(<module>)
        1    0.291    0.291    5.149    5.149 <ipython-input-4-9421bc2016cb>:13(mandel_set)
  1000000    3.461    0.000    4.849    0.000 <ipython-input-4-9421bc2016cb>:5(mandel)
 24214592    1.388    0.000    1.388    0.000 {built-in method builtins.abs}
        1    0.008    0.008    0.008    0.008 <ipython-input-4-9421bc2016cb>:17(<listcomp>)
        2    0.000    0.000    0.000    0.000 <ipython-input-4-9421bc2016cb>:1(linspace)
        2    0.000    0.000    0.000    0.000 <ipython-input-4-9421bc2016cb>:3(<listcomp>)
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
```

PROFILING A PYTHON CODE: LINE LEVEL

- Use the `line_profiler` package

```
[In] %load_ext line_profiler
[In] %lprun -f mandel mandel_set()
```

```
Timer unit: 1e-06 s
Total time: 12.4456 s
File: <ipython-input-2-9421bc2016cb>
Function: mandel at line 5
#Line       Hits         Time  Per Hit   % Time  Line Contents
==============================================================
     5                                           def mandel(c, maxiter):
     6    1000000     250304.0      0.3      1.1      z = c
     7   24463110    6337732.0      0.3     27.7      for n in range(maxiter):
     8   24214592    8327289.0      0.3     36.5          if abs(z) > 2:
     9     751482     201108.0      0.3      0.9              return n
    10   23463110    7658255.0      0.3     33.5          z = z*z + c
    11     248518      65444.0      0.3      0.3      return n
```

PROFILING A PYTHON CODE: LINE LEVEL

This can be done in console mode as well

```
@profile
def mandel(c, maxiter):
  z = c
  for n in range(maxiter):
    if abs(z) >2:
      return n
    z = z*z +c
  return n
```

Then on the command line

```
kernprof -l -v mandel.py
```

Then

```
python3 -m line_profiler mandel.py.lprof
```

PROFILING A PYTHON CODE: MEMORY

- Use the `memory_profiler` package

```
[In] %load_ext memory_profiler
[In] %mprun -f mandel mandel_set()
```

```
Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
     8    118.2 MiB -39057.7 MiB     1000000   def mandel(c, maxiter):
     9    118.2 MiB -39175.5 MiB     1000000       z = c
    10    118.2 MiB -293081.8 MiB    24463110      for n in range(maxiter):
    11    118.2 MiB -292425.7 MiB    24214592        if abs(z) > 2:
    12    118.2 MiB -38519.6 MiB       751482          return n
    13    118.2 MiB -253906.1 MiB    23463110        z = z*z + c
    14    118.2 MiB   -656.4 MiB      248518      return n
```

PROFILING A PYTHON CODE: MEMORY

- Use the `memory_profiler` package

```
@profile
def mandel(c, maxiter):
    z = c
    for n in range(maxiter):
        if abs(z) >2:
            return n
        z = z*z +c
    return n
```

Then on the command line

```
mprof run mandel.py
```

Then

```
mprof plot
```

Or

```
python3 -m memory_profiler mandel.py
```