Profiling and Optimising in Python: CHEAT SHEET

Carpentries incubator lesson:
https://carpentries-incubator.github.io/pando-python/

Basics

As code grows in complexity, it starts taking longer to execute. Improving code's performance will makes the work easier and saves energy.

The Carpentries Incubator lesson can be found on the GitHub repository: **https://github.com/carpentries-incubator/pando-python**

Profiling

The process of measuring the performance of the code to identify which parts of the code are taking the most time to run.

Not every script needs profiling: For example, for a one-off script that runs quickly, profiling will not be necessary. However, for a long-running script that is expected to be run multiple times, profiling can be very useful.

Types of profiling

Manual profiling
Use the `time` module to measure the time taken by different sections of the code.

Function-level profiling
Measures time taken by individual functions in the code.

Line-level profiling
Measures the time taken by individual lines of code.

Timeline profiling
Gives an idea of the execution of the code over time.

Hardware metric profiling
Measures hardware related metrics.

Function-level profiling

Measures the time taken by individual functions in your code., to help identify which functions are taking the most time to execute.

cProfile
Commonly used tool for function-level profiling. It is a part of the Python standard library.

Run

```
python -m cProfile my_script.py
```

Store output

```
python -m cProfile -o output.prof my_script.py
```

Visualise output using `snakeviz`

```
snakeviz output.prof
```

Line-level profiling

Records the time taken to execute individual lines of code.

line_profiler
Commonly used tool for line-level profiling. It is not a part of the Python standard library and needs to be installed using:

```
pip install "line_profiler[all]"
```

Decorate functions to be profiled

```
from line_profiler import LineProfiler
@profile
def my_function(arg):
  ...
  return
print(my_function(arg=arg_value))
```

Run

```
python -m kernprof -lvr my_script.py
```

Output
Outputs to a file (my_script.py.lprof). This file is not human-readable. To print it to console:

```
python -m line_profiler -rm my_script.py.lprof
```

Optimisation

The process of improving the performance of your code by making changes to it.

Potential risks associated with optimising:

- Performing optimisations in a way that make code harder to understand.
- Changing the code with a potential to introduce new bugs.
- Trying to optimise code that only takes up a minuscule fraction of the total runtime, you might end up spending more time on the optimisation than you actually gain from it.

Simple tips to optimise Python code

- Use built-in functions and the standard library - don't reinvent the wheel!
- Use list comprehension when creating lists
- Prefer tuples, dictionaries, and sets over lists, when appropriate
- Minimise Python written: Use scientific Python packages (NumPy, Pandas). NumPy arrays support broadcasting (mathematical operation is automatically parallelised over elements of the array - much faster than explicit for-loop)
- Newer is often faster: When possible, use the latest versions of Python and packages
- One large file is better than many small files
- Avoid destroying and recreating objects

Link to the lesson