# Practice 4.1

Practice sheets are not assessed. The intention is to use material from lectures in preparation for Competence tests and Assignments. You are encouraged to use `thing.[TAB]` and `thing?` in IPython.

## Multiprocessing

The following imports the `time` function from the `time` module. (Yes, it's annoying that these have the same name.)

```
from time import time
```

The `time` function returns the number of seconds since the UNIX epoch, Thursday 1st January 1970 UTC.

```
time()
```

```
1544088756.7157552
```

This can be used to see how long something takes to run. e.g.

```
def factors(N):
    return [n for n in range(2,N//2) if N%n == 0]

start_time = time()
numbers = factors(10000000)
end_time = time()
duration = end_time - start_time
duration
```

```
0.2684800624847412
```

Use `multiprocessing` to factor the following numbers and record how long it takes.

```
Ns = [327668137, 293041039, 302952547, 316772299, 323579981, 286725589, 313867091, 360613859,
      307593911, 334270493, 327857639, 369562007, 344286581, 288065551, 313064867, 306248237]
```

You can adapt the following code snippet:

```
from multiprocessing import Pool

def f(n):
    return n**2

if __name__=='__main__': # this is enouraged in general and necessary on Windows
    with Pool() as p:
        out = p.map(f, range(10))
```

(For a discussion of `__name__` etc see e.g. here or docs.)

Compare this with the time taken *without* using multiprocessing. Can you make it go any faster?

## Stability

Plot the energy $E = x^2 + v^2$, as a function of time, for the undamped harmonic oscillator $\ddot{x} + x = 0$ starting with $\dot{x} = 0$ and $x = 1$ and $0 < t < 20$, using Euler, Runge–Kutta 4 (pure Python), and Runge–Kutta 6 (from `scipy`), for different time-steps $\delta t = 10^{-4}, 10^{-3}, 10^{-2}, 5 \times 10^{-2}$.

How does the error grow for each method?

How long does each method take with each $\delta t$?