KIT103/KMA155 Practical 07: Finding primes and factorisation

This week we'll implement the sieve of Eratosthenes and use the primes generated to test other numbers for primality and to derive factorisations.

Preparation

Start Spyder and save the temp.py file in the editor with a new name as you will save your work in it.

1 Sieving for primes

> **Task:** Implement both versions of the prime generating functions from the lecture slides (note that if you choose to copy the text from the slides you will need to do some manual reformatting).
>
> Test that they both work correctly (i.e., they produce a list or set of prime numbers).

1.1 How much faster is the sieve?

In this task you will time how long each prime-generating function takes to run, using a more flexible version of the timing function introduced in the last practical. You may wish to skip it for now and try it later on. Add the following import to your source file:

```
import time
```

Then define this function, which will time the execution of one of your given prime-generating functions:

```
def time_primes(pf, n, show_msg=True):
    start = time.clock()
    total = len(pf(n))
    elapsed = time.clock() - start
    if show_msg:
        print('Took {0:g} seconds to produce the {1} primes up to
{2}'.format(elapsed, total, n))
    return elapsed
```

Call `time_primes` with the name of one of your prime-generating functions, the value of *n* and (optionally) `False` if you want to suppress the message. Try running it in a loop with values of *n* from [10, 100, 1000, ..., 100000].

2 Is it prime?

Task: Implement a function called `is_prime` that takes a single integer `n` and returns `True` iff `n` is prime (or rather, `n` is not a compound number). The algorithm should be as follows:

- Generate prime numbers up to no more than the square root of `n`; if using the more efficient `primes2` then call `sorted()` on its result to get a sorted *list* of prime numbers
- Until you have exhausted the list of primes, test if the current prime number `p` evenly divides `n` (i.e., if the remainder [modulo] of dividing `n` by `p` is zero); if so then return `False` immediately
- If you reach the end of the list of primes then `n` is prime (so return `True`)

2.1 Factorising compound numbers

Task: Implement the prime factors algorithm described in the lecture slides. There are many minor variations in how this can be done. The first step can be the same as in the `is_prime` function you defined earlier, but generate primes up to `n`.