> *Then you must cut down the mightiest tree in the forest... with... a herring!*

Loop Invariants

It is not always easy to tell if a loop will create the correct result in every circumstance. Therefore, we use a **loop invariant** that states if this condition is True then the result should be correct. For this to work completely the invariant needs to be True at the start, at the end of the loop body and if the loop condition is False the invariant must still be True.

Here is an example that finds the prime factors of a number and puts them into a list. This function will use the `next_prime` function found in prime.py.

```
from prime import *

def factorise(n) :
    """Returns a list of the prime factors of 'n'.

    Parameters:
        n (int): Number for which factors are to be found.

    Preconditions:
        n >= 2

    Return:
        list<int>: Prime factors of 'n'.
    """
    prime = 2
    factors = []
    while n > 1 :
        # Loop Invariant: product(factors) * n == initial n
        while n % prime == 0 :
            factors.append(prime)
            n /= prime
        prime = next_prime(prime)
    return factors
```

This function starts with the current prime number set to 2, as 2 is the first prime number, and an empty list to store the primes in. We then start a while loop that keeps iterating as long as n > 1. We do not want to go less than 2 as there are no primes less than 2. It is here, in a comment, that we introduce our loop invariant. In this case it is that the product of all the primes found so far multiplied by the current number that we are operating on equals the initial number. We then start another while loop that keeps appending the current prime number to the factors list while the current number is divisible by that prime, also dividing the current number by that prime to move onto the next number. Back in the first while loop we use the `next_prime` function to move onto the next prime number. The last thing the function does is return the list of factors.

Saving as factorise.py we can run a few tests to show this function working.

```
>>> factorise(10)
[2, 5]
>>> factorise(340)
[2, 2, 5, 17]
>>> 2 * 2 * 5 * 17
340
```