Explanation to the first hundred problems from Project Euler with Python 3

Max Halford

Project Euler is a good way to learn basic number theory, to get your imagination going and to learn a new programming language. If you can solve the first hundred problems then you can solve any problem, as long as you keep being curious and you use your imagination, personally I decided to work on other styles of projects, there isn't just number theory out there! However solving number theory problems is a good way to learn a programming language : you need to be rigorous and tidy. Most solutions require smart algorithms and not brute force approaches. Google is your best friend when you know what to do but you don't know how to write it, or when you don't understand the code you're reading. I will not babysit the reader but to the contrary assume that he knows how to google "Python ¡insert command¿", I am not being harsh : knowing how to find documentation when coding is of the utmost importance because you mostly have to teach yourself notions, you have to be self-educated. I would recommend approaching challenging problems with pen and paper and maybe some mathematical research. The association between human ingenuity and the computational power of our machines can produce wonderful results, as long as one doesn't lean on the latter. Python provides many coding styles and paradigms, googling "PEP 8" and "Google Python Style Guide" will make pick up good habits early on. Keep your code simple, it has to be readable by everyone. Assign comprehensible names to your variables and parameters. Don't reinvent the wheel, the Python community is very large and modules have been written for a lot of things, use them. Internet isn't always right, for example one liners are not a good thing : they are difficult to read afterwards. In a perfect world reading your code should feel like reading a book, keep that in mind. Don't be frightened to go on internetand find the solutions to problems, it's actually counterproductive to search for answers for too long, time is precious and there are too many things to cover. The main objective is to learn, not to be proud of yourself.

1 Multiples of 3 and 5

A question that often comes up with Python is "What is the best way to go through a list?". Problematically there are many answers and it all comes down to personal preferences. The two main choices are Map-Reduce-Filter-Lambda (http://www.python-course.eu/lambda.php) and list comprehensions (http://www.python-course.eu/list_comprehension.php). I would suggest reading up on both approaches, it is always good to know what tools are at your disposal without having to be a black belt at them. I mostly use list comprehensions, they are, in my opinion, more comprehensible and flexible. The problem is straightforward with a list comprehension : build a list bounded by 1 and 999 with elements being divible by 3 or 5 and sum up all the elements.

2 Even Fibonacci numbers

When you read a problem for the first time do as much research as you need on the topics of the problem. In this case Fibonacci numbers are relatively famous but will find some rather less famous concepts later that will require some looking around. After some pondering one notices that a number in the Fibonacci sequence is only defined by the two numbers preceding it. What results from this observation is that it is useless to store the sequence, but instead we should go through it and pick the numbers we want (in this case they have to be even). We use the property of the sequence in the following algorithm :

- Let α and β be adjacent numbers in the sequence.
- β' becomes the following number in the sequence : $\alpha + \beta$.
- α' becomes β.

3 Largest prime factor

It is a well known theorem that any number can be written as the product of prime numbers. Hence any number divided by all its factors successively will return 1. So what we can do is, for a given integer n, go through every integer k inferior to it and superior to 1 and check if $n \bmod k = 0$. If it isn't then increment k and try again. When it is then k is a factor of n, so we store it and start over the same process for n/k. The neat thing with this

algorithm is that you do not have to check if k is prime. Indeed k will be checked before $2k, 3k, 4k$ etcetera.

4 Largest palindrome product

The easiest way to check if a number, or any string for that matter, is a palindrome, is to compare it with its reverse. Python is good for string manipulation, I would suggest reading http://www.pythoncentral.io/how-to-slice-listsarrays-and-tuples-in-python/ to learn useful dodges instead of reinventing the wheel. Now that we have a function to check if a number is palindrome, we need to apply it on every product of two 3-digit numbers in the descending order (because we are looking for the biggest palindrome), hence the iteration through $\{999..100\} \times \{999..100\}$. Notice how easily a two dimensional matrix is defined in one line with a list comprehension.

5 Smallest multiple

Recursion is a concept that can not be avoided when programming, I would suggest googling it if you are not familiar with it. It is easy enough to check for a given integer n from 1 till k (in our case 20), we simply go through the list and check the divisibility. Now we have to find an integer that satisfies the previous algorithm for $k = 20$. A brute force approach is to iterate every integer one by one, but we quickly realize that the algorithm takes for ever. It takes some insight to notice that the integer n that verifies the algorithm for k is a factor of the integer m, hence when looking for m we can increment by n instead of 1. However efficient this insight is, it is not intuitive, try some cases on a piece of paper. An example that comes to mind is this : 6 is divisible by 1, 2 and 3. 24 is divisible by 1, 2, 3 and 4. 6 is a factor of 24 hence we only look for the candidates 12, 18 and 24 when searching for the smallest integer divisible by 1, 2, 3 and 4.