

Project Euler

Write-ups on Project Euler problems

- [Euler 0001](#)
- [Euler 0002](#)
- [Euler 0003](#)
- [Euler 0004](#)

Euler 0001

Problem:

Listing all multiples of 3 and 5 under 1000.

Given: 3,5,6,9 are all multiples of 3 and 5 below 10.

Considerations:

We could iterate through every multiple of these numbers but there is going to be overlap when their multiples are divisible by both of them. Such examples would be 15 or 30. In these cases we don't want to add these numbers to the totals twice. This is what we need to watch out for in the code.

Naive approach:

1. Keep iterating through all the multiples of 3 and 5
2. Add each number to a set, or a list with a not in conditional
3. Sum the set or list

```
target_under = 1000
multiples = [3,5]
found = []

#go through each of our multiples
for multiple in multiples:
    #set the running total equal to the first multiple
    running_total = multiple
    #keep incrementing until we go over the target, then go to the next multiple
    while running_total < target_under:
        #add the running total to the list if it is not already in it.
        if running_total not in found:
            found += [running_total]
        running_total += multiple
```

Euler 0002

Problem

Each new term in the Fibonacci sequence is generated by adding the previous two terms. By starting with 1 and 2, the first terms will be: 1,2,3,5,8,13,21,34,55,89

By considering the terms in the Fibonacci sequence whose values do not exceed four million, find the sum of the even-valued terms.

What we know:

- We need to generate the Fibonacci sequence up to 4,000,000.
- We only need to keep every other number in the sequence.

Considerations

Time-space considerations and decisions to be made:

- We could choose to store 0 numbers and recursively generate all numbers up to 4,000,000.
 - This spends both a lot of time although it doesn't use a large permanent space.
- We could generate all of the Fib. numbers up to 4,000,000 and store them in a list, good for referencing later, and will reduce redundancy.
- We could keep only the last 2 Fib. numbers and keep dropping the last one when we get a new one.

```
fib_1 = 1
fib_2 = 2
limit = 4000000

#running total set to 2 because it is odd
running_total = 2

#keep going until fib_1 and fib_2 totaled > limit
while fib_1 + fib_2 < limit:
    fib = fib_1 + fib_2
    fib_1 = fib_2
    fib_2 = fib

    #add to the running total if it is even.
```

```
if fib % 2 == 0:  
    running_total += fib
```

Euler 0003

The problem:

The prime factors of 13195 are 5, 7, 13, and 29.

What is the largest prime factor of the number 600951475143?

The considerations:

There are a lot of approaches, of which I will probably use later.

We are going to start with some basic sieving and then work out a more general algorithm:

Iterate from 2 to the upper limit:

- Is it divisible by 2:
 - add 2 to the prime factors list
 - set the loop to iterate up to $\text{upper_limit}/2$
 - Is it divisible by 2:
 - Keep doing above operation
- Is it divisible by 3:
 - add 3 to the prime factors list
 - set the loop to iterate up to $\text{upper_limit}/3$
- We don't care about 4.
- Is it divisible by 5:
 - add 5 to the prime factors list
 - set the loop to iterate up to $\text{upper_limit}/5$
- We don't care about 6.
- Is it divisible by 7:
 - add 5 to the prime factors list
 - set the loop to iterate up to $\text{upper_limit}/7$

etc.

Something to note: We don't care about any multiples of 2 and 3 after we check them, so a quick way to get around checking these multiples we can iterate through the loop by adding 6 and checking the before and after numbers.

```
upper_limit = 6009515658416
#upper_limit = 20023110541 #, good to use to double check. Is 2 [2,2],3,167
current_upper_limit = upper_limit
prime_factors = []

#we are going to hardcode 2 and 3 for looping sake
```

```

if upper_limit % 2 == 0:
    while current_upper_limit % 2 == 0:
        current_upper_limit = current_upper_limit/2
        prime_factors += [2]
if upper_limit % 3 == 0:
    while current_upper_limit % 3 == 0:
        current_upper_limit = current_upper_limit/3
        prime_factors += [3]
current = 6

#go up to the upper limit of numbers, add 2 for edge case (I'm not sure that part is correct)
while current < current_upper_limit + 2:
    #print(prime_factors, current_upper_limit)
    #print(prime_factors, current_upper_limit)
    #check the numbers before and after the increment of 6
    #add them to the prime factors as many times as necessary
    #keep dividing the new number to use.
    if current_upper_limit % (current - 1) == 0:
        while current_upper_limit % (current - 1) == 0:
            current_upper_limit = current_upper_limit/(current-1)
            prime_factors += [current-1]
    if current_upper_limit % (current + 1) == 0:
        while current_upper_limit % (current + 1) == 0:
            current_upper_limit = current_upper_limit/(current+1)
            prime_factors += [current+1]
    current += 6

print(prime_factors)

```

The twist:

While I was testing this it took insanely long because I was calculating for 600951475143 instead of 600851475143.

The largest prime factor of $600 \times 8 \times 51475143$ is 6857. This is not very high, and easy enough to brute force.

The largest prime factor of $600 \times 9 \times 51475143$ is 1552846189. This is a lot of iterations to get too and the code would timeout.

After reading the solution provided by Project Euler I understood what I was missing to get the solution to this typo'd part.

Every number can have at most one prime factor that is greater than its square root.
I have modified the code below to reflect this change and break out of the while loop if the `current_upper_limit` is greater than the square root of the original number.

This means that we only have to iterate through 775210 (square root of 600951475143) before we (can rightfully) call it quits and accept that:

- This is prime
- This is the last prime factor

```
import math

upper_limit = 600951565841652
#upper_limit = 20023110541 #, good to use to double check. Is [2,2,3,167]
current_upper_limit = upper_limit
sqrt_limit = math.sqrt(upper_limit)
prime_factors = []

#we are going to hardcode 2 and 3 for looping sake
if upper_limit % 2 == 0:
    while current_upper_limit % 2 == 0:
        current_upper_limit = current_upper_limit/2
        prime_factors += [2]
if upper_limit % 3 == 0:
    while current_upper_limit % 3 == 0:
        current_upper_limit = current_upper_limit/3
        prime_factors += [3]
current = 6

#go up to the square root
while current < sqrt_limit:
    #print(prime_factors, current_upper_limit)
    #check the numbers before and after the increment of 6
    #add them to the prime factors as many times as necessary
    #keep dividing the new number to use.
    if current_upper_limit % (current - 1) == 0:
        while current_upper_limit % (current - 1) == 0:
            current_upper_limit = current_upper_limit/(current-1)
            prime_factors += [current-1]
    if current_upper_limit % (current + 1) == 0:
        while current_upper_limit % (current + 1) == 0:
            current_upper_limit = current_upper_limit/(current+1)
```

```
        prime_factors += [current+1]
    current += 6

    #if we looped beyond the square root limit and the number
    #we are left with isn't 1, then it is the final prime factor.
    if current_upper_limit != 1:
        prime_factors += [current_upper_limit]

    print(prime_factors)
```


Euler 0004

The Problem:

9009 is a palindromic number since it can be read forwards and backwards the same. It is the product of 91×99 and is the largest palindromic number generated by the product of 2 2-digit numbers.

What is the largest palindromic number that is the product of 2 3-digit numbers?

The approach:

The incredibly naive approach would be to start with 999×999 and iterate through all iterations going down through all the numbers $999 \rightarrow 1$ and $999 \rightarrow 1$. We are probably going to find the number pretty near the top, but still running the entire brute force is *only* 998001 iterations... which isn't *too* bad.

As for the checker, we can convert the number into a string and check if it the same forwards and backwards.

The code:

```
largest_palindrome = 0

def palindrome(number):
    return str(number) == str(number)[::-1]

for i in range(1000,1,-1):
    for j in range(1000,1,-1):
        if palindrome(i*j) and i*j > largest_palindrome:
            largest_palindrome = i*j

print(largest_palindrome)
```