Leap Year Checker

This code takes a year as an input, turns it into a int type variable and return if the year has an extra day in it or not.

1. Original Code:

```
3   def is_leap(year):
4       leap = False
5       if year % 4 == 0:
6           leap = True
7           if year % 100 == 0:
8               leap = False
9               if year % 400 == 0:
10                  leap = True
11
12      return leap
```

This code uses a simple and straightforward logic to determine if a year is a leap year or not. It checks if the year is divisible by 4, and then checks if it's divisible by 100 and 400 to determine if it's a leap year. The code is easy to read and understand, but it's a bit verbose and has some nested if statements that can make it hard to follow.

2. Alternate Code:

```
17  def vers2(year):
18      if year % 4 != 0:
19          return False
20      elif year % 100 != 0:
21          return True
22      elif year % 400 != 0:
23          return False
24      else:
25          return True
26
```

This code is similar to the original code, but it doesn't use a `leap` variable to keep track of whether the year is a leap year or not. The code is easy to read and understand.

3. Optimized Code:

```
14    def minu_vers(year):
15        return year & 3 == 0 and (year % 100 != 0 or year & 400 == 0)
```

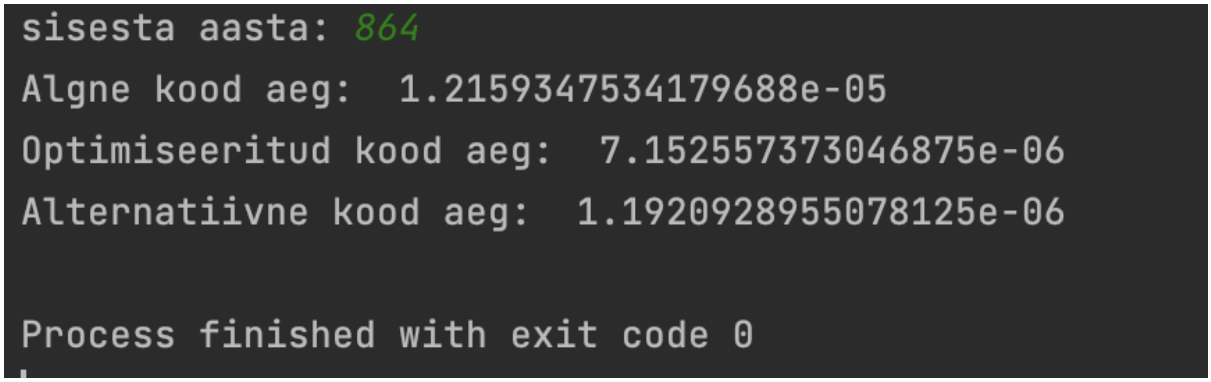
This code uses a more concise logic to determine if a year is a leap year or not. It checks if the year is divisible by 4, and then checks if it's not divisible by 100 or if it's divisible by 400 to determine if it's a leap year. The code is shorter and easier to read than the original code, but it can be a bit harder to understand for someone who is not familiar with boolean logic or bitwise operations.

In terms of speed and efficiency, the optimized code is the fastest of the three because it uses a simple and concise logic that involves fewer operations. The other two codes are also relatively fast and efficient, but they are slightly slower than the optimized code because they involve more operations or more nested `if` statements.

In terms of reliability, all three codes are reliable and should work correctly for most use cases. However, the optimized code may be slightly more reliable than the other two codes because it uses a simpler logic that is less prone to errors or bugs.

Overall, the optimized code is the best choice because it's fast, efficient, and easy to read and understand. However, the other two codes are also good alternatives if you prefer a more verbose or explicit coding style.

But we have to keep in mind that that is not always the case. There were tests were alternative code was faster than the optimised one. Yet I didn't find a year were the original code was the fastest.



Analysed by Ott-Kaarel Vään and Matthias Kalevi Paluteder