Daily Coding Problem #202

Problem

This problem was asked by Palantir.

Write a program that checks whether an integer is a palindrome. For example, 121 is a palindrome, as well as 888. 678 is not a palindrome. Do not convert the integer into a string.

Solution

Without the restriction on using strings, this problem would be trivial:

```
def check_palindrome(num):
    return str(num) == ''.join(reversed(str(num)))
```

Taking an idea from the above solution, we can instead try to *mathematically* create the reversed integer, and check to see that the original and reversed numbers are the same. How might we do this?

Recall that because we use the decimal system, any n-digit integer $x_1x_2...x_n$ can be represented as $x_1 * 10^{n-1} + x_2 * 10^{n-2} + ... + x_n * 10^0$.

As a result, we can get the digits in reverse order by repeatedly taking the number mod 10 and dividing by 10. For example, if we start with the number 678, we would do the following:

- Take `678 % 10` as the first digit, then divide by `10`
- Take `67 % 10` as the second digit, then divide by `10`
- Take `6 % 10` as the third digit

Then, if we want to build the reversed number, we can repeatedly add each digit to 10 times our current sum. So to obtain 876, we would do the following:

- Begin with 8
- Multiply `8 * 10`, then add our new number 7
- Multiply `87 * 10`, then add our new number 6

We can combine these steps as follows:

```
def check_palindrome(num):
    tmp = num
    reversed_num = 0

    while tmp != 0:
        reversed_num = (reversed_num * 10) + (tmp % 10)
        tmp /= 10

    return num == reversed_num
```

This algorithm runs in O(N) time, where N is the number of digits in our input, and uses some extra space to store the reversed number. What if we weren't allowed that extra space?

One way to do this is as follows: we begin with the first and last digit of our number. If the digits match, we create a new number by removing those digits.

We continue in this way, comparing first and last digits and stripping them from our number, until either there is a mismatch or there are no digits left. If we end up without any digits left, we'll know that we have a palindrome.

The tricky part about this is figuring out how to get and remove the first and last digits mathematically. Let n be the number of digits in our integer. Then we can perform these operations as follows:

- Get the first digit: `int // 10`^{n}
- Get the last digit: `n % 10`
- Remove the first digit: `int % 10`^{n}
- Remove the last digit: `n // 10`

Putting this all together, we have the following algorithm:

```
def check_palindrome(num):
    # Find the divisor needed to obtain the first digit.
    divisor = 1
    while (num / divisor >= 10):
        divisor *= 10

    while (num != 0):
        first = num / divisor
        last = num % 10

        if (first != last):
            return False

        num = (num % divisor) / 10

        # The number of digits has been reduced by 2, so our divisor is reduced
by 10 ** 2.
        divisor = divisor / 100

    return True
```