Jeremy Thaller

Q: A car's odometer reads 12345.6 mi and it's trip-meter reads 123.4 mi. How many miles must you drive until the first four digits of the odometer and trip-meter match again?

First multiply both trip and odometer by 10...

```
In[184]:= a = 1234;
          b = 123456;
          i = 1;
```

This loop iterates until the first 4 digits of each number are equal. The Floor function throws out anything after the decimal point. (e.g. Floor[25.98]=25). The bizarre denominator limits the function to be 4 orders of magnitude. To understand why, consider what $\log_{10}(x)$ means. $\text{Log}_{10}(x)$ means, "10 to what power equals x?" So $\log_{10}(100)$= 2, and $\log_{10}(589)$ = 2 point something. The point being, taking the Floor of the base-10 log of any number returns one less than the number of digits in that number.

So, for both numbers, we only want the first 4 digits. For a=1234, if we take Floor[Log_{10}(a)], we'll get 3. But a starts with 4 digits, so we want to keep all of them. Thus, we need to divide by 1 and take the Floor. To correct for this, instead of dividing by 10^3, we need to subtract 3 from the exponent so we divide by 10^0=1 instead. For the b case, we have b=123,456. So taking $\log_{10}(b)$ = 5. We're starting with a 6 digit number, and want to keep the first 4. Thus we need to divide by 100 and take the floor of it. As you can see, this also needs the correction factor of 10^{-3}.

The while loop compares the first 4 digits of a and b. If they don't match, it adds 1 to a and b and checks again, iterating and iterating until they match.

```
In[48]:= While[Floor[(a + i)/10^(Floor[Log10[a+i]-3])] ≠ Floor[(b + i)/10^(Floor[Log10[b+i]-3])], i++];
         i
Out[49]= 12336
```

Divide by 10 again... Note: //N just forces the number to output as a decimal instead of a fraction.

```
Out[7]= 1233.6
```

```
In[42]:= Mod[Floor[(a + i)/10^(Floor[Log10[a+i]-3])], 10^4]
Out[42]= 1357
```

```
In[38]:= Mod[Floor[(b + i)/10^(Floor[Log10[b+i]-3])], 10^6]
Out[38]= 1357
```

```
In[10]:= a / 10 + i // N
Out[10]= 1357.
```

```
In[11]:= b / 10 + i // N
Out[11]= 13 579.2
```

So the odometer reads 13,579.2 mi and the trip-meter reads 1357.0 mi

```
In[13]:= a + 10 101
Out[13]= 11 335

In[14]:= b + 10 101
Out[14]= 133 557
```

Part 2:

Q: What if the numbers reset such that a can only be 4 digits long and b can only be 6 digits long?

Now that a resets to 0 instead of becoming 5 digits, we can just account for this by using the modulo operator. a mod n is the remainder of the division of a by n. Thus, if we use “a mod 10^4”, anything for a < 10^4 is simply a, and anything for a > 10^4 is going to be what remains. Because of this, we don’t need to take into account the strange denominator. However, b is more complicated. This time, b resets to 0 when it hits 7 digits, so we need to use b mod 10^6. Additionally, we’re still only interesting in the first four digits of b, so we still need to include the denominator of $10^{\text{Floor}[\text{Log10}[b+i]-3]}$ and take the floor of this value (for the same reasons as in part 1).

```
In[60]:= i = 1;
         While[Mod[a + i, 10^4] ≠ Floor[Mod[b + i, 10^6] / 10^(Floor[Log10[b+i]-3])], i++];
         i
Out[62]= 10 101

In[58]:= a / 10 + i / 10 // N
Out[58]= 1133.5

In[59]:= b / 10 + i / 10 // N
Out[59]= 13 355.7
```

Thus, after 1,010.1 mi, the odometer will read 13,355.7 and the trip-meter will read 1133.5. It should be noted that 1,010.1 is palindromic.

Part 3:

If the odometer reads 12345.6 and the trip meter reads 123.4, how far do you need to travel until the number 0-9 all show up between the two meters? Assume that the counters have limited digits and reset to zero, as in part 2.

My strategy will just be to compare each digit to all the others until they are unique and keep iterating as before.

First, we need to find a way to extract each digit. I’ll do this by taking moduli. If I were doing this in another language, I would probably convert to strings and just extract the chars, even if it’s less efficient.

For a

```
In[327]:= Floor[Mod[1234, 10] ](*gives last digit from left*)
          Floor[(Mod[1234, 100] - Mod[1234, 10]) / 10 ](*gives 3rd digit from left*)
          Floor[(Mod[1234, 1000] - Mod[1234, 100]) / 100 ](*gives 2nd digit from left*)
          Floor[(Mod[1234, 10000] - Mod[1234, 1000]) / 1000](*gives 1st digit from left*)

Out[327]= 4

Out[328]= 3

Out[329]= 2

Out[330]= 1
```

For b

```
In[316]:= Floor[Mod[Mod[123456, 10], 10^6]](*gives last digit from left*)
          Floor[(Mod[123456, 100] - Mod[123456, 10]) / 10 ]
          (*gives second to last digit from left*)
          Floor[(Mod[123456, 1000] - Mod[123456, 100]) / 100 ]
          (*gives third to last digit from left*)
          Floor[(Mod[123456, 10000] - Mod[123456, 1000]) / 1000 ]
          (*gives third digit from left*)
          Floor[(Mod[123456, 100000] - Mod[123456, 10000]) / 10000 ]
          (*gives second digit from left*)
          Floor[(Mod[123456, 1000000] - Mod[123456, 100000]) / 100000]
          (*gives first digit from left*)

Out[316]= 6

Out[317]= 5

Out[318]= 4

Out[319]= 3

Out[320]= 2

Out[321]= 1
```

Okay, so here’s the giant disgusting function that solves it. It works by using a nested for-loop. It picks a value of i (used to increase a and b), and then checking if the numbers 0 through 9 appear in either a or b. Since there are only 10 digits total and 10 numbers, we know that each number must appear once, and thus, if any number appears twice, than another number between [0,9] must be missing. Note that in Mathematica, for loops functions as thus: For[*start, test, increment, body*]. For loops work by executing *start*, then repeatedly evaluating *body* and *incr* until *test* fails to return True. Thus for the inner for loop, I checked to see if the numbers 0 showed up in a+i or b+i, If is does, it checks if we’ve checked for all numbers 0-9. If we haven’t, we just iterate and check the next number. If we have checked 0-9 and it hasn’t broken yet, the we know it’s a solution and we append the value of i (distance traveled) to our list of solutions. Note, in the inner loop, if any number doesn’t show up, then the inner loop breaks, and the outer loop iterates by increasing the value of i.

In[335]:=
```
c = False;
Solutions = {};
For[i = 1, Length[Solutions] < 10, i++,
  For[j = 0, (j == Floor[(Mod[a + i, 10000] - Mod[a + i, 1000])/1000] ||
      j == Floor[1/100 (Mod[a + i, 1000] - Mod[a + i, 100])] ||
      j == Floor[1/10 (Mod[a + i, 100] - Mod[a + i, 10])] || j == Mod[a + i, 10] ||
      j == Floor[Mod[b + i, 10]] || j == Floor[1/10 (Mod[b + i, 100] - Mod[b + i, 10])] ||
      j == Floor[1/100 (Mod[b + i, 1000] - Mod[b + i, 100])] ||
      j == Floor[(Mod[b + i, 10000] - Mod[b + i, 1000])/1000] ||
      j == Floor[(Mod[b + i, 100000] - Mod[b + i, 10000])/10000] ||
      j == Floor[(Mod[b + i, 1000000] - Mod[b + i, 100000])/100000]),
    j++, If[(j == Floor[(Mod[a + i, 10000] - Mod[a + i, 1000])/1000] ||
        j == Floor[1/100 (Mod[a + i, 1000] - Mod[a + i, 100])] ||
        j == Floor[1/10 (Mod[a + i, 100] - Mod[a + i, 10])] || j == Mod[a + i, 10] ||
        j == Floor[Mod[b + i, 10]] || j == Floor[1/10 (Mod[b + i, 100] - Mod[b + i, 10])] ||
        j == Floor[1/100 (Mod[b + i, 1000] - Mod[b + i, 100])] ||
        j == Floor[(Mod[b + i, 10000] - Mod[b + i, 1000])/1000] ||
        j == Floor[(Mod[b + i, 100000] - Mod[b + i, 10000])/10000] ||
        j == Floor[(Mod[b + i, 1000000] - Mod[b + i, 100000])/100000]) &&
      j == 9, AppendTo[Solutions, i]] ] ];
i
```

Out[338]=
```
14449
```

Print the first 10 solutions:

In[326]:=
```
For[k = 0, k < 10, Print[Solutions[[k]]], k++]
```

```
8611

8620

11 334

11 451

11 613

11 640

13 053

13 494

14 034

14 448
```

Check if first solution works:

```
In[258]:= a + 8611

Out[258]= 9845

In[259]:= b + 8611

Out[259]= 132 067
```

And the second:

```
In[272]:= a + 8620

Out[272]= 9854

In[273]:= b + 8620

Out[273]= 132 076
```

And the third? Note, because of how it wraps you can throw the first digit of a out.

```
In[277]:= a + 11 334

Out[277]= 12 568

In[278]:= b + 11 334

Out[278]= 134 790
```