Advent of Code 2025

Jump to problem

Day 1: Secret Entrance

Today was fairly difficult as far as Day 1's go (or maybe I was tired from the 4 hour drive through a rainstorm I had just hours earlier). As in the past, I did the problems in Kotlin.

Part 1

I started part 1 off wonderfully, misreading it, and (attempting to) solve part 2. When I was getting a higher answer than expected, I went back, reread, and stashed my code for later, (correctly) assuming it may be useful for part 2. Other than that, my code wasn't that interesting, just a check for zero, and repeatedly incrementing my `dial` variable by 100 until it was positive, since modulo arithmetic wouldn't help me there.

Total time: **00:06:03**

Part 2

Part 2 is where I really began to struggle. At first, I attempted the "smarter" solution. It looked something like this (unfortunately my 12:30am brain decided not to save it for when I want to come back later to fix it):

```
dial += value * direction
if (dial == 0) {
    zeroCounter++
}

while (dial < 100) {
    dial += 100
    zeroCounter++
}

while (dial > 100) {
    dial -= 100
    zeroCounter++
}
```

The main issue (at least as far as I could tell at the time) was my code would "double count" the dial if it was initially zero before an instruction, and moved to the left. I added a check to cover this case, but it didn't seem to work. This solution is definitely one I would like to revisit after the calendar is over, as it seems like one that could be optimized heavily.

After this attempt, I moved onto the "brute-force" method. I ran a loop from 1 to the instruction's "value", counting every time the dial passed zero, and ensuring it stayed within the range reqired. This ended up working, and giving me the correct solution after many incorrect submissions.

Total time: **00:38:10**

Day 2: Gift Shop

Overall today was way easier than day 1 in my opinion. Most of my pitfalls were due more to me misreading the problem than anything else.

Part 1

My initial solution to this completely ignored everything except the first and last in the range. For some reason, I completely missed that the input was a range and not two unrelated numbers. This wasn't that large a change however, since I was already using `flatMap` to parse the input (just returning a list of the first and second numbers in the list).

To actually process the numbers, I initially used Kotlin's `take` and `drop` methods, roughly as follows:

```
val first = id.take(id.length / 2)
val second = id.drop(id.length / 2)

if (first == second) count += id.toInt()
```

This worked for Part 1, but would later be changed after I introduced a helper function in Part 2.

Total time: **00:05:49**

Part 2

For this part, I quickly realized a helper function to "chunkify" a string and check the uniqueness of these "chunks" would be helpful, allowing me to refactor Part 1 as well. Luckily, Kotlin provides a `String.chunk` method that does just that. To check for a repeating pattern, I simply turned the chunks into a set, and checked if its length was 1, indicating there was only 1 unique "chunk":

```
fun checkRepeats(string: String, num: Int): Boolean {
    val chunks = string.chunked(num)

    return chunks.toSet().size == 1
}
```

After writing this function, all I needed to change from my part 1 code was loop from 1 to half the length of each id (any further and there was no possibility for a repeating pattern), checking for a repeating pattern iff the current size evenly divides the id's size (another minor optimization, albeit important for size=1 on part 1, where ids such as `111` would be mistakenly verified since 1, the first "half" of the string, would cause the remainder to be split into thirds instead of halves).

Total time: **00:14:32**