Day 11

Part 1

The ancient civilization on Pluto was known for its ability to manipulate spacetime, and while The Historians explore their infinite corridors, you've noticed a strange set of physics-defying stones.

At first glance, they seem like normal stones: they're arranged in a perfectly straight line, and each stone has a number engraved on it.

The strange part is that every time you blink, the stones change.

Sometimes, the number engraved on a stone changes. Other times, a stone might split in two, causing all the other stones to shift over a bit to make room in their perfectly straight line.

As you observe them for a while, you find that the stones have a consistent behavior. Every time you blink, the stones each simultaneously change according to the first applicable rule in this list:

- If the stone is engraved with the number `0`, it is replaced by a stone engraved with the number `1`.
- If the stone is engraved with a number that has an even number of digits, it is replaced by two stones. The left half of the digits are engraved on the new left stone, and the right half of the digits are engraved on the new right stone. (The new numbers don't keep extra leading zeroes: `1000` would become stones `10` and `0`.)
- If none of the other rules apply, the stone is replaced by a new stone; the old stone's number multiplied by `2024` is engraved on the new stone.

No matter how the stones change, their order is preserved, and they stay on their perfectly straight line.

How will the stones evolve if you keep blinking at them? You take a note of the number engraved on each stone in the line (your puzzle input).

If you have an arrangement of five stones engraved with the numbers `0 1 10 99 999` and you blink once, the stones transform as follows:

- The first stone, `0`, becomes a stone marked `1`.
- The second stone, `1`, is multiplied by `2024` to become `2024`.
- The third stone, `10`, is split into a stone marked `1` followed by a stone marked `0`.
- The fourth stone, `99`, is split into two stones marked `9`.
- The fifth stone, `999`, is replaced by a stone marked `2021976`.

So, after blinking once, your five stones would become an arrangement of seven stones engraved with the numbers 1 `2024 1 0 9 9 2021976`.

Here is a longer example:

Initial arrangement: `125 17`

After 1 blink: `253000 1 7`

After 2 blinks: `253 0 2024 14168`

After 3 blinks: `512072 1 20 24 28676032`

After 4 blinks: `512 72 2024 2 0 2 4 2867 6032`

After 5 blinks: `1036288 7 2 20 24 4048 1 4048 8096 28 67 60 32`

After 6 blinks: `2097446912 14168 4048 2 0 2 4 40 48 2024 40 48 80 96 2 8 6 7 6 0 3 2`

In this example, after blinking six times, you would have 22 stones. After blinking 25 times, you would have `55312` stones!

Consider the arrangement of stones in front of you. How many stones will you have after blinking 25 times?

In [1]:
```
from collections import defaultdict
from pathlib import Path
```

In [2]:
```
test_path = Path("data/day11_test.txt")
data_path = Path("data/day11_data.txt")

def load_data(fpath: Path) -> list[str]:
    """Returns data as list of strings."""
    return fpath.open().read().strip().split()


# I thought I was being clever/preparing for scaling by caching
# those values. Reader, I was not.
def blink(stones: list[str], cache: dict[str:str]={"0": ["1"]}) -> tuple[list[str], d
    """Carries out a single blink on a row of stones."""
    newstones = []  # Holds new stones after blink

    for stone in stones:  # Calculate the post-blink stone
        if stone in cache:  # Use cached next value
            newstone = cache[stone]
        else:
            if len(stone) % 2 == 0:
                newstone = [str(int(_)) for _ in [stone[:len(stone)//2],
                                                  stone[len(stone)//2:]]]
            else:
                newstone = [str(int(stone) * 2024)]
            cache[stone] = newstone
        newstones += newstone

    # Return the stones after the blink, with the cache of next values
    return newstones[:], cache


def multiblink(stones: list[str], n: int, cache: dict[str, str]={"0": ["1"]}):
    """Returns the row of stones after a number of blinks."""
    for idx in range(n):
        stones, cache = blink(stones, cache)

    return stones[:]


test_stones = ["125", "17"]
for blinks in (6, 25):
    print(len(multiblink(test_stones, blinks)))
```

```
22
55312
```

In [3]:
```
stones = load_data(data_path)
len(multiblink(stones, 25))
```

Out[3]: 202019

Part 2

The Historians sure are taking a long time. To be fair, the infinite corridors are very large.

How many stones would you have after blinking a total of `75` times?

In [4]:
```
# Reader, I can't tell you how long it took me to get this
def stoneblinks(stones: list[str], blinks: int) -> int:
    """Returns the number of stones after the passed amount of blinks."""
    # Store the stones as a dictionary of the counts of each individual
    # stone number in the row
    stonedict = defaultdict(int)
    for stone in stones:
        stonedict[stone] += 1

    # We'll now update the counts of the stones with each individual number
    # rather than maintaining a list or recursing down the scads of updates.
    for depth in range(blinks):
        newstones = defaultdict(int)  # The stones in the next row

        for stone in stonedict:  # Update the next row counts for each current stone
            if stone == "0":
                newstones["1"] += stonedict[stone]
            elif len(stone) % 2 == 0:
                newstones[str(int(stone[:len(stone)//2]))] += stonedict[stone]
                newstones[str(int(stone[len(stone)//2:]))] += stonedict[stone]
            else:
                newstones[str(int(stone) * 2024)] += stonedict[stone]

        stonedict = newstones  # The current row is now the new row

    # Return the sum of counts
    return sum([v for k, v in stonedict.items()])


stoneblinks(["125", "17"], 6)
```

Out[4]: 22

In [5]:
```
stones = load_data(data_path)
stoneblinks(stones, 75)
```

Out[5]: 239321955280205

In [ ]:
```
```