🕯️ Advent of Code 2025 😀

Contents

day 1

Problem summary

Given a **sequence** of dial **rotations** on a **circular scale from 0 to 99**, **starting at** position **50**, determine **how often** the dial **reaches position 0** after applying each rotation in order.

Mathematical formulation

- Let the dial positions be elements of the cyclic group $\mathbb{Z}_{100}$.
- Let the initial position be $x_0 = 50$.
- Let the sequence of n rotations be (d_i, k_i) with $d_i \in \{L, R\}$ and $k_i \in \mathbb{N}$.

Define the signed rotation

$$s_i = \begin{cases} -k_i \text{ if } d_i = L \\ k_i \text{ if } d_i = R \end{cases}$$

and the position update

$$x_i \equiv x_{i-1} + s_i \pmod{100}, \quad i = 1, \ldots, n.$$

The password is the number of indices i for which the dial reaches zero

$$P = |\{i \in \{1, \ldots, n\} : x_i = 0\}|.$$

Python3

python

```
>>> xs = '\nL68\nL30\nR48\nL5\nR60\nL55\nL1\nL99\nR14\nL82\n'
>>>
>>> temp, count = 50, 0
>>>
>>> list( map(int, xs.replace("L", "-").replace("R", "+").split())  )
[-68, -30, 48, -5, 60, -55, -1, -99, 14, -82]
  ↳ Define the signed rotation
                                    s_i = { -k_i if d_i = L
                                          {  k_i if d_i = R
>>>
>>> for r in map(int, xs.replace("L", "-").replace("R", "+").split()):
...     temp = (temp + r) % 100
  ↳ and the position update
                                    x_i ≡ x_{i-1} + s_i (mod 100),   i = 1, ..., n.
...     if temp == 0: count += 1
  ↳ The password is the number of indices i for which the dial reaches zero
                                    P = |{i ∈ {1, ..., n} : x_i = 0}|.
...
>>> print(count)
3
>>>
>>> (lambda xs: (
...     lambda pos, c:
...         ([(pos := (pos + r) % 100,
...            c := c + (pos == 0))
...           for r in map(int, xs.replace("L", "-").replace("R", "+").split())],
...          c)[1]
... )(50, 0))(xs)
3
```

typst

typst

```
#let day1(xs) = {
  let steps = xs.replace("L", "-").replace("R", "+").split().map(int)
  let result = steps.fold((50, 0), (acc, r) => {
    let pos = calc.rem((acc.at(0) + r), 100)
    let cnt = acc.at(1) + int(pos == 0)
    (pos, cnt)
  })
  result.at(1)
}
```

typst

```
#day1("L68 L30 R48 L5 R60 L55 L1 L99 R14 L82")
```

3

Docs

day1

Day 1: Secret Entrance

$(\text{L68}, \text{L30}, \text{R48}, \text{L5}, \text{R60}, \text{L55}, \text{L1}, \text{L99}, \text{R14}, \text{L82})^{\intercal} \to 3$

```
#day1("L68 L30 R48 L5 R60 L55 L1
L99 R14 L82")
```

3

Parameters

day1(xs: string) -> int

xs string

The argument

Part Two

py

```
>>> for r in map(int, xs.replace("L", "-").replace("R", "+").split()):
...     temp = (temp + r) % 100
...     if temp == 0: count += 1
```

↧

py

```
>>> temp, count = 50, 0
>>>
>>> for r in map(int, xs.replace("L", "-").replace("R", "+").split()):
...     if r ⩾ 0:
...         for _ in range(r):
...             temp = (temp + 1) % 100
...             if temp == 0: count += 1
...     else:
...         for _ in range(r * (-1)):
...             temp = (temp - 1) % 100
...             if temp == 0: count += 1
...
>>> print(temp, count)
28 6
```

⇓

py

```
for r in map(int, xs.replace("L", "-").replace("R", "+").split()):
    for _ in range(abs(r)):
        temp = (temp + (1 if r > 0 else -1)) % 100
        count += temp == 0
```

Day 2: Gift Shop

Where $\mathcal{R}$ is the Set of Ranges R, and id $\in R$ being a product identifier. The solution is:

$$\sum_{R \in \mathcal{R}} \sum_{\text{id} \in R} \begin{cases} \text{id if id is valid} \\ 0 \text{ otherwise} \end{cases}$$

py

```py
from day2 import xs

ds = [
  range(
    *(lambda ys: [ys[0], ys[1] + 1])(
      list(map(int, e.split("-")))
    )
  )
  for e in xs.split(",")
]


def p(x: int) -> int:
  x = str(x)

  if len(x) % 2 != 0: return 0

  return 0 if x[:len(x)//2] == x[len(x)//2:] else int(x)

sum(sum(map(p, d)) for d in ds)
```

Part Two

Now the Predicate changes for whether a product id is valid or not ...

> Now, an **ID is invalid if** it is made only of some sequence of **digits repeated at least twice**. So, 12341234 (1234 two times), 123123123 (123 three times), 1212121212 (12 five times), and 1111111 (1 seven times) are all invalid IDs.

Algorithm 1: PredicateForValidProductIDs

input $n \in \mathbb{N}$ is the ID that should be checked
$l \leftarrow \lfloor \log_{10}(n) \rfloor + 1 \quad \forall n > 0$ is the `len(str(n))`
for each $i \in \left[\frac{l}{2}\right]$ where [n] is {1, ..., n}
  head ← first i chars of string representation of n
  if repeat(head, n=l / i) == str(n) **then**
    return $\bot$
  end
end
return $\top$

Algorithm 1: some caption

⇓

py

```py
def p(x: int) -> int:
  x = str(x)
  l = len(x)

  for i in range(l // 2):
    h = x[:i+1]
    if h * (l // (i+1)) == x:
      return int(x)

  return 0
```

Day 3: Lobby

python

```python

xs = """987654321111111
811111111111119
234234234234278
818181911112111"""

xs = [list(map(int, e)) for e in xs.split()]


def day3(xs: list[list[int]]) -> int:
  def helper(x: list[int]) -> int:
    temp: int = 0

    for i, v in enumerate(x):
         for ii, e in enumerate(x[i+1:]):
             if int(str(v) + str(e)) > temp:
                 temp = int(str(v) + str(e))

    return temp

  return sum(
    helper(x)
    for x in xs
  )

print(f"{ day3(xs) = }")
```

Part Tow

> The task per row is: Given a sequence of digits, select exactly 12 digits, in order, such that the resulting 12 digit number is maximized.

py

```python

from itertools import combinations


sum(map(
  lambda y: max(map(
    lambda x: int("".join([str(e) for e in x])),
    list(combinations(y, 12))
  )),
  xs
))
```

this is very slow for the big input …

⇓

py

```python
print(f"[INFO] { len(xs) = }, { len(xs[0]) = }")


s: int = 0


from tqdm import tqdm

for index, y in enumerate(xs):
  temp: int = 0

  print(f"[INFO] { index = }")

  for x in tqdm(combinations(y, 12)):
    g = int("".join( map(str, x)))

    if g > temp:
      temp = g

  s += temp

  print(f"{temp = }")

print(f"final { s = }")

```

```
[INFO]  len(xs) = 200,  len(xs[0]) = 100
[INFO]  index = 0
??????????it [0?:??, ~700000.00it/s]^C
Traceback (most recent call last):
  File "/…/day3.py", line 258, in <module>
    g = int("".join( map(str, x)))
        ^^^^^^^^^^^^^^^^^^^^^^^^^^
KeyboardInterrupt
```

$$\frac{\binom{100}{12}\ \text{it}}{700000\frac{\text{it}}{\text{sec}}} \Rightarrow\approx 1.501 \times 10^{9}\ \text{sec} \Rightarrow 47.55\ \text{average Gregorian years …}$$

This shit takes way to long …

⇓

Process digits from left to right, maintaining a stack:

- You may delete at most $n - k$ digits.
- While the current digit is larger than the last chosen digit, and deletions remain, remove the last digit.
- Append the current digit.
- At the end, truncate to length k.

This is optimal and runs in **linear time**

py

```python
xs = """987654321111111
811111111111119
234234234234278
818181911112111"""

xs = [list(map(int, e)) for e in xs.split()]

def max_subsequence_value(x: list[int], k: int = 12) -> int:
    drop = len(x) - k
    stack: list[int] = []

    for d in x:
        while drop > 0 and stack and stack[-1] < d:
            stack.pop()
            drop -= 1
        stack.append(d)

    stack = stack[:k]

    val = 0
    for d in stack:
        val = val * 10 + d
    return val


def day3(xs: list[list[int]]) -> int:
    return sum(max_subsequence_value(x, 12) for x in xs)

print(f"{day3(xs) = }")
```

runs in $\approx 0.02\,\text{sec}$ for the big input 😊