AoC Day 13

Puzzle background

- Trying to win prizes on claw machines that have two buttons
  - Claw starts at (0, 0)
  - Prize is at known location (x, y)
  - First button costs 3 tokens to push, adds offsets to x and y
  - Second button costs 1 token to push, adds different offsets to x and y

```
Button A: X+94, Y+34
Button B: X+22, Y+67
Prize: X=8400, Y=5400

Solution: 80*94+40*22=8400, 80*34+40*67=5400
```

Winning this machine costs `80*3+40*1=280` tokens

Naive solution

- For part 1, we have a reasonable upper limit on button pushes (100 each)
- Simple: iterate through all possible combinations, find solutions
  - Requires 100*100=10,000 checks per claw machine
- Can extract variables using regex

Part 2 always get you

- It’s fun to guess how part 2 will mess you up
- In this case, it’s just big numbers
- Targets are now around 10,000,000,000,000
- Average offset is 50
- We’d need an average of (10,000,000,000,000^2)/50 button pushes
- That’s 2,000,000,000,000,000,000,000,000
- No, thanks

But hey, it’s just systems of linear equations

- Most AoC puzzles just require having a key insight

```
m*94+n*34=8400
m*22+n*67=5400
```

This gives us two equations to solve for two free variables (m and n), which we could do with pen and paper for fun

$$94a + 22b = 8400$$
$$34a + 67b = 5400$$

$$34(94a + 22b) = 34*8400$$
$$94(34a + 67b) = 94*5400$$

$$3196a + 748b = 285600$$
$$3196a + 6298b = 507600$$

$$5550b = 222000$$
$$b = 40$$

$$94a + 22*40 = 8400$$
$$94a = 8400 - 880$$
$$94a = 7520$$
$$a = 80$$

We could build a solution solver

- A lot of work
- Could be fun
- We're lazy, tho, so we use linear algebra

$$\begin{bmatrix} 94 & 22 \\ 34 & 67 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} 8400 \\ 5400 \end{bmatrix}, Ax = B$$

$$x = A^{-1}B, \text{ where}$$

$$\text{For } A = \begin{bmatrix} p & q \\ r & s \end{bmatrix}, A^{-1} = \frac{1}{(p \cdot s) - (q \cdot r)} \begin{bmatrix} s & -q \\ -r & p \end{bmatrix}$$

Even lazier

- numpy can do that using the np.linalg.solve function

```
import numpy as np
A = [[94, 22], [34, 67]]
b = [8400, 5400]
x = np.linalg.solve(A, b)

A = [[26, 67], [66, 21]]
b = [12748, 12176]
x = np.linalg.solve(A, b)
```

Using sympy

- https://www.sympy.org/en/index.html
- “SymPy is a Python library for symbolic mathematics”
- Can just write equations in close-to-human notation and have it solve them

```
from sympy import symbols, Eq, solve
n, m = symbols("n m", integer=True)
eq1 = Eq(xs[0] * n + xs[1] * m, xs[2])
eq2 = Eq(ys[0] * n + ys[1] * m, ys[2])
solution = solve((eq1, eq2), (n, m))
if solution:
    part1 += solution[n] * 3 + solution[m]
```