```
using PlutoUI
```

```
parse_file (generic function with 1 method)
```

```
function parse_file(io::IO)
    line = readline(io)
    return split(line, ",") .|> n -> parse(Int8, n)
end
```

Problem 1

```
simulate_lanternfish (generic function with 1 method)
```

```
function simulate_lanternfish(initial_state, number_of_days)
    simulated_state = copy(initial_state)

    while number_of_days > 0
        # Removed the initial optimization I thought of because
        # in practice it was approx. simulating everyday. So
        # removing it helped to remove the extra cost of finding min
        # making it faster
        days_to_simulate = 1 #min(min(simulated_state...)+1, number_of_days)

        simulated_state .-= days_to_simulate
        new_fish_to_generate = count(n -> n < 0, simulated_state)
        simulated_state[simulated_state .< 0] .= 6

        push!(simulated_state, repeat([8], new_fish_to_generate)...)

        number_of_days -= days_to_simulate
    end

    return length(simulated_state)
end
```

```
358214
```

```
  2.053307 seconds (7.05 M allocations: 399.826 MiB, 3.32% gc time, 97.34% compilation
```

```
with_terminal() do
    open("./Day6/prob_input.txt") do io
        initial_state = parse_file(io)
        @time simulate_lanternfish(initial_state, 80)
    end
end
```

Problem 2

The previous solution was unoptimized as it was using arrays which kept on getting bigger and bigger and it also had to iterate over the evergrowing array on everyday to simulate. Instead of that brute force approach we optimized it by computing state before hand and simulating over that.

What we do is that as we know the life of a fish can be maximum `8` so we only need to maintain number of fishes who have life `[0,1,2,3,4,5,6,7,8]` and then we can just move the number of fished back and add new fishes. E.g. if there are 113 fished who have life span of 3 then we just -113 from state that is keeping track of lifespan of 3 and add $+113$ to the state keeping track of lifespan of 2.

```
simulate_lanternfish_optim (generic function with 1 method)
```

```
function simulate_lanternfish_optim(initial_state, number_of_days)
    NEW_FISH_LIFE = 8
    REJUVINATE_LIFE = 6
    computed_state = zeros(Int128, NEW_FISH_LIFE+1)
    for fish_life in initial_state
        # Plus 1 is because Julia is 1-based index
        computed_state[fish_life + 1] += 1
    end

    for _ in 1:number_of_days
        fish_recycled = computed_state[1]
        computed_state[1:NEW_FISH_LIFE] .= computed_state[2:NEW_FISH_LIFE+1]
        computed_state[NEW_FISH_LIFE + 1] = fish_recycled
        computed_state[REJUVINATE_LIFE + 1] += fish_recycled
    end

    return sum(computed_state)
end
```

```
1622533344325
```

```
  0.000020 seconds (257 allocations: 52.219 KiB)
```

```
with_terminal() do
    open("./Day6/prob_input.txt") do io
        initial_state = parse_file(io)
        @time simulate_lanternfish_optim(initial_state, 256)
    end
end
```