day6

December 6, 2020

0.1 Day 6 - Customs Declarations

This feels fairly easy at first glance, this is all about building sets. Again, we are after seperating the input by blank lines, which was a bit of a pain back in a few earlier days

[1]:
```
import ipytest
ipytest.autoconfig()
```

[2]:
```
lines = [line.strip() for line in open("day6.txt").readlines()]
lines2 = ["$$" if (line == "") else line for line in lines]
lines3 = "".join(lines2).split('$$')
```

By joining all the lines together, we're loosing individual answers, but we probably don't care, since we are considering people as groups instead.

So now all we need to do is turn the line into a set, which will discard dup0licates and then count the lengths of teh sets, and sum them all…

[3]:
```
sum([len(set(l)) for l in lines3])
```

[3]:
```
6885
```

0.1.1 Part 2

Darn it, I knew I needed to keep track of individuals. In this case we're going to need to parse a bit better. The basic premise is similar, but now we need produce a set per group, that set will be the intersection of all of the sets of answers. Then we can simply count the sets again…

So if we had rewritten the previous version as:

`groups = [set(l) for l in lines3] sum = sum([len(group) for group in groups]`

It would be clear that the only real difference here is that our groups assignment is more complex…

[4]:
```
lines = [line.strip() for line in open("day6.txt").readlines()] + [""]
# We now have each group ending in a single ""

groups = []
group = None
for line in lines:
    if line == "":
        groups.append(group)
        group = None
    else:
        if group == None:
            group = set(line)
        else:
            group &= set(line)

sum([len(group) for group in groups])
```

[4]:
```
3550
```

0.2 Refactoring and making nicer

Ok, I did this in record time, so I wonder if I can make this a bit cleaner, with code that based on the previous 6 days, I might need to reuse.

0.2.1 Part 1 - Inputs in groups

We've had several inputs now where it's multiple lines, separated by an empty line. My current solution is both nasty, uses a weird sentinel, and loses the line breaks, which often have meaning. Let's do something better that can take the following input: ‘abc de

acd d

a a a‘

and turn it into `[ ['abc', 'de'], ['acd', 'd], ['a', 'a', 'a'] ]`

[5]:
```
def parse_groups(lines):
    result = []
    group = []
    for line in lines:
        if line == "":
            result.append(group)
            group = []
        else:
            group.append(line)
    result.append(group)
    return result

assert [['a', 'b']] == parse_groups(["a", "b"])
assert [['a', 'b'], ['c']] == parse_groups(['a', 'b', '', 'c'])

def parse_groups_from_file(filename):
    return parse_groups([l.strip() for l in open(filename)])
```

Now we can refactor our answers for 1 and 2.

Additionally, I didn't like the special code to handle the first line in the set, so I'm going to try using what is effectively the infinite set, for our input at least

```
[6]: groups = parse_groups_from_file('day6.txt')
     groupsets = [set("".join(l)) for l in groups]
     print(sum([len(group) for group in groupsets]))

     group_intersections = []
     for group in groups:
         groupset = set("abcdefghijklmnopqrstuvwxyz")
         for person in group:
             groupset &= set(person)
         group_intersections.append(groupset)
     print(sum([len(group) for group in group_intersections]))
```

```
6885
3550
```

That's much nicer for part 2, but that looks an awful lot like a reduce function on each set, I wonder if functools can make that even clearer

```
[7]: import functools
     infiniteset = set("abcdefghijklmnopqrstuvwxyz")
     group_intersections = [functools.reduce(lambda accumulator,person: accumulator␣
      ↪& set(person), group, infiniteset) for group in groups]
     print(sum([len(group) for group in group_intersections]))
```

```
3550
```

There we go. I'm not actually sure how much clearer that is, the lambda somewhat loses the fact that the groups are lists of lists, and that we're turning each person in the group into a set, and then intersecting the sets. But it works, and is much fewer lines of code