day4

December 6, 2020

0.1 Day 4 - The Passport Scanner

So we have to read a set of passport data and validate whether there are any missing fields. We can do this the hard way or the easy way.

The hard way involves setting up a data structure, filling it in as we parse the fields, and then validating that all values have been filled in.

The slightly easier way is to recognise in this case that the passport scanner doesn't care about the values, and is instead just interested in the key names. We can simply hold a set or list of field titles, remove them as we see them and validate that the set is empty or contains cid.

Undoubtably, the next step will require rewriting it if we do this, but lets give it a try

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

[2]:
```
def is_valid_passport(line):
    expected = set(["byr", "iyr", "eyr", "hgt", "hcl", "ecl", "pid", "cid"])
    seen = set(["cid"])
    for kv in line.split():
        token = kv.split(":")[0]
        seen.add(token)
    return seen==expected

def count_passports(lines):
    count = 0
    for line in lines:
        if is_valid_passport(line):
            count += 1
    return count
```

[3]:
```
assert is_valid_passport("ecl:gry pid:860033327 eyr:2020 hcl:#fffffd byr:1937␣
↪iyr:2017 cid:147 hgt:183cm")
assert not is_valid_passport("iyr:2013 ecl:amb cid:350 eyr:2023 pid:028048884␣
↪hcl:#cfa07d byr:1929")
assert is_valid_passport("hcl:#ae17e1 iyr:2013 eyr:2024 ecl:brn pid:760753108␣
↪byr:1931 hgt:179cm")
```

```
assert not is_valid_passport("hcl:#cfa07d eyr:2025 pid:166559648 iyr:2011 ecl:
 ↪brn hgt:59in")

test1 = ["ecl:gry pid:860033327 eyr:2020 hcl:#fffffd byr:1937 iyr:2017 cid:147␣
 ↪hgt:183cm",
" iyr:2013 ecl:amb cid:350 eyr:2023 pid:028048884 hcl:#cfa07d byr:1929 ",
" hcl:#ae17e1 iyr:2013 cid:350 eyr:2024 ecl:brn pid:760753108 byr:1931 hgt:
 ↪179cm",
"hcl:#cfa07d eyr:2025 pid:166559648 iyr:2011 ecl:brn hgt:59in "]

assert count_passports(test1) == 2
```

Ok, so that seems to work, so lets try on real data.

First we need to get one passport per line. I can't think of a better way of munging the text than looking for the delimeters (empty strings) and then splitting on those

[4]:
```
lines = [line.strip() for line in open("day4.txt").readlines()]
lines2 = ["$$" if (line == "") else line for line in lines]
lines3 = " ".join(lines2).split('$$')
print(len(lines3))
print(count_passports(lines3))
```

```
291
235
```

0.2 Part 2 Ignoring values bites us

Sigh, we now need to validate the passport data as well. That's where our first option would have worked better.

But never fear, we've got at least a single process for validating a single line of passport, so we can look at the field, and validate it at that point.

What we can do is quick return False if the field doesn't validate, and use some form of validator based on the field info. At least we don't need to reference any other data, so it's simple enough.

[5]:
```
import re
hcl = re.compile("#[0-9a-f]{6}")
pid = re.compile("\d{9}")
hgt = re.compile("\d{2,3}(cm|in)")
def validate_field(field, data):
    if field == "byr":
        idata = int(data)
        return 1920 <= idata <= 2002
    if field == "iyr":
        idata = int(data)
        return 2010 <= idata <= 2020
    if field == "eyr":
```

```
        idata = int(data)
        return 2020 <= idata <= 2030
    if field == "hgt":
        if not hgt.fullmatch(data): return False
        idata = int(data[:-2])
        if data[-2:] == "cm":
            return 150 <= idata <= 193
        if data[-2:] == "in":
            return 59 <= idata <= 76
    if field == "hcl":
        return hcl.fullmatch(data) != None
    if field == "ecl":
        return data in ["amb", "blu", "brn", "gry", "grn", "hzl", "oth"]
    if field == "pid":
        return pid.fullmatch(data) != None
    if field == "cid":
        return True
    return False


assert validate_field("byr", "2002")
assert validate_field("iyr", "2010")
assert validate_field("eyr", "2020")
assert validate_field("hgt", "179cm")
assert validate_field("hgt", "76in")
assert validate_field("hcl", "#fe123a")
assert validate_field("hcl", "#123abc")
assert not validate_field("hcl", "#z12345")
assert validate_field("ecl", "brn")
assert validate_field("pid", "012345678")
assert validate_field("pid", "000000000")
assert not validate_field("pid", "01234567")
```

Now we can try rewriting is_valid_passport to use that

[6]:
```
def is_valid_passport(line):
    expected = set(["byr", "iyr", "eyr", "hgt", "hcl", "ecl", "pid", "cid"])
    seen = set(["cid"])
    for kv in line.split():
        token,value = kv.split(":")
        if validate_field(token, value):
            seen.add(token)
        else:
            return False
    return seen==expected
```

```
assert is_valid_passport("pid:087499704 hgt:74in ecl:grn iyr:2012 eyr:2030 byr:
↪1980 hcl:#623a2f")
assert is_valid_passport("eyr:2029 ecl:blu cid:129 byr:1989 iyr:2014 pid:
↪896056539 hcl:#a97842 hgt:165cm")
assert is_valid_passport("hcl:#888785 hgt:164cm byr:2001 iyr:2015 cid:88 pid:
↪545766238 ecl:hzl eyr:2022")
assert is_valid_passport("iyr:2010 hgt:158cm hcl:#b6652a ecl:blu byr:1944 eyr:
↪2021 pid:093154719")

assert not is_valid_passport("eyr:1972 cid:100 hcl:#18171d ecl:amb hgt:170 pid:
↪186cm iyr:2018 byr:1926")
assert not is_valid_passport("iyr:2019 hcl:#602927 eyr:1967 hgt:170cm ecl:grn␣
↪pid:012533040 byr:1946")
assert not is_valid_passport("hcl:dab227 iyr:2012 ecl:brn hgt:182cm pid:
↪021572410 eyr:2020 byr:1992 cid:277")
assert not is_valid_passport("hgt:59cm ecl:zzz eyr:2038 hcl:74454a iyr:2023 pid:
↪3556412378 byr:2007")
```

Great, lets try on the real data!

[7]:
```
print(len(lines3))
print(count_passports(lines3))
```

```
291
194
```