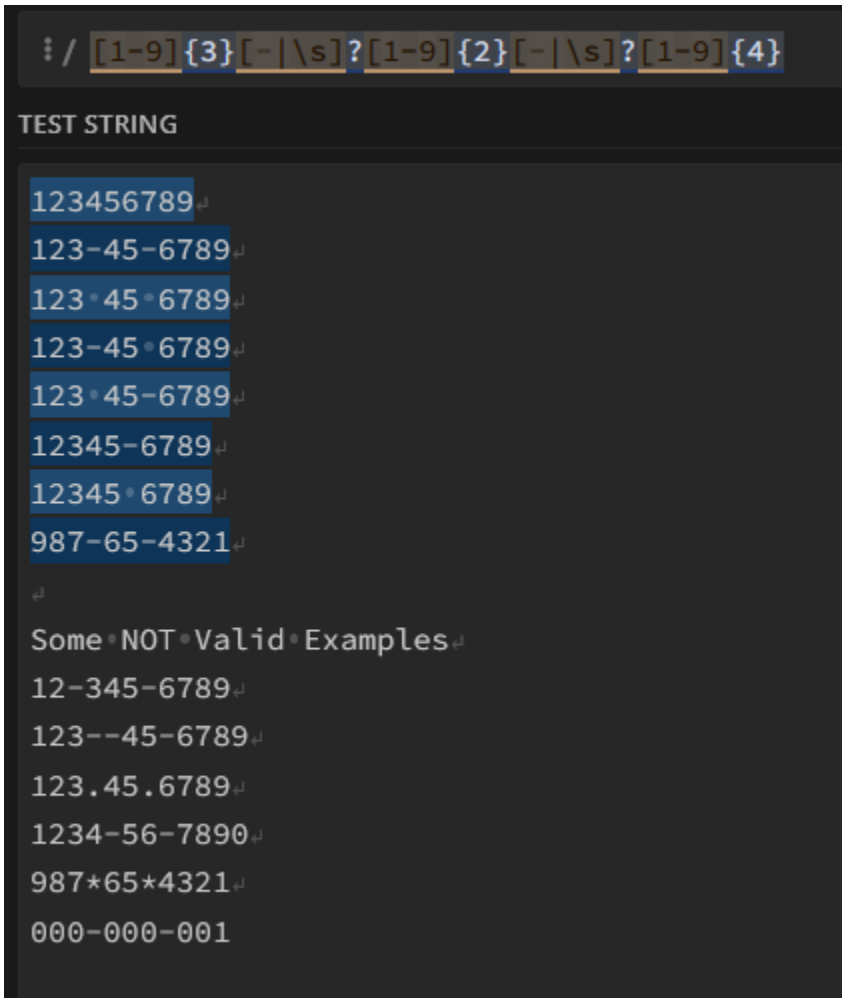
1. Social Security Number
    a. [1-9]{3}[-|\s]?[1-9]{2}[-|\s]?[1-9]{4}

    b. For this regular expression I wanted to be able to support the social security number format with spaces or dashes or nothing, I also wanted to support any number 1 through 9. I did not think that there could be 0 in a ssn. I used the brackets after every number to make sure that the pattern of number in a ssn was enforced. I used or to make sure that there was a dash or a space but not both but also made it optional.

    c. For testing I used regex101.com I took some of the examples from the doc as well as adding my own to test the formatting was enforced and only spaces or dashed were used

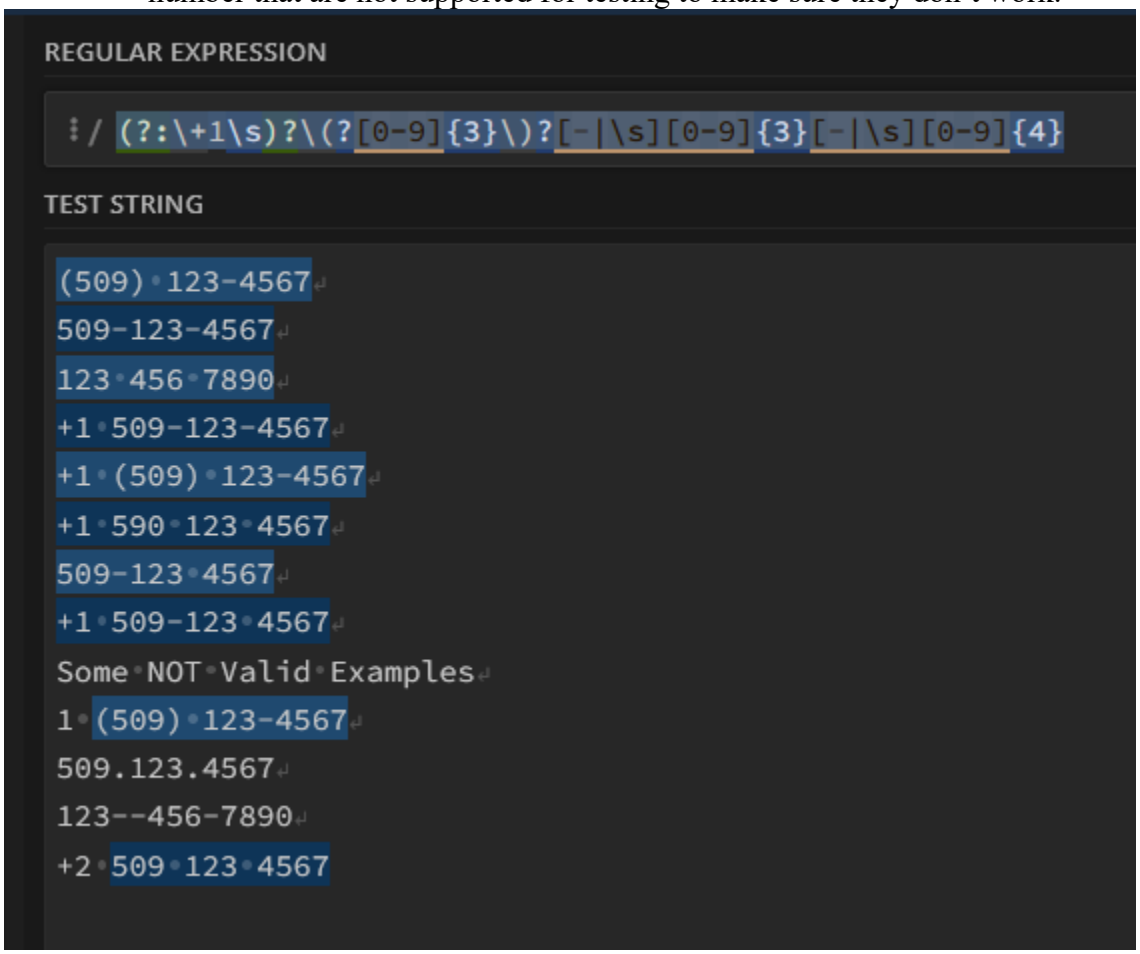


2. US Phone Number
    a. (?:\+1\s)?\(?[0-9]{3}\)?[-|\s][0-9]{3}[-|\s][0-9]{4}

    b. For this regex expression I wanted to support us phone numbers, enforcing the format while offering flexibility for any US phone number. Starting with (?:)? I

am saying everything in here must match if it is in the string. In the parentheses is +1\s, so if a phone number starts with a + it must be followed by a 1 and a space. I wanted to support phone numbers that contained a 0 so I used the in-range block to make sure that only numbers 0-9 were read. I also used brackets to enforce the number patterns of a phone number in the 3 total blocks of 3, 3, and 4 digits. I also enforced needing some type of separator between the blocks. It has to be a dash or a space and nothing else.

c. For testing I used regex101.com. I used a few examples from the doc and wrote a few of my own. I wanted to make sure that any number between 0-9 was supported while enforcing the formatting of a phone number. I also included a few number that are not supported for testing to make sure they don't work.

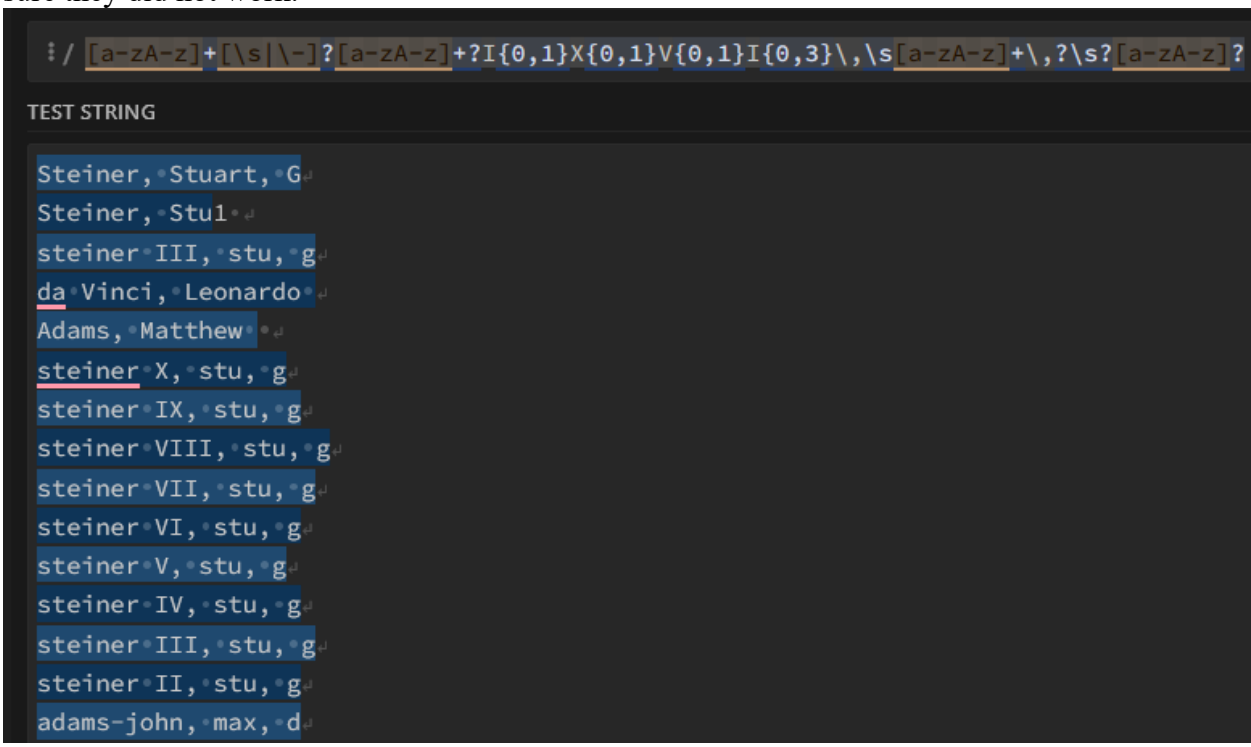


3. Name On Class Roster
   a. [a-zA-z]+[\s|\-]?[a-zA-z]+?I{0,1}X{0,1}V{0,1}I{0,3}\,\s[a-zA-z]+\,?\s?[a-zA-z]?

   b. For this regular expression I wanted to be able to support a name of any size while strictly enforcing the format, but also providing enough flexibility to support last names with spaces or dashes (-). To enforce this formatting of Last name (optional roman numerals), First name, MI (optional) I started by using [a-zA-z] + this lets me read in any word character one or more times (I tried using \w but it was

reading in numbers), that means the name can be any length if it is a word. Next, I have a space or a dash that are optional followed by a char in [a-zA-z] once or more. After that is the roman numeral. I only supported up to ten to allow flexibility while keeping it relatively simple. I used the brackets to keep them optional but to also keep the formatting of roman numerals. If I had used * it would have recognized VIIIIIII which is not correct roman numerals. This is then followed by a non-optional comma space and a char in [a-zA-z] once or more to enforce a first name. It is then followed by an optional comma, space, and a char in [a-zA-z] to enforce a 1 char middle enital.

c. To test these I used regex101.com and some names that matched the formatting, had 2 last names or a dash, and roman numerals from one to ten, and an optional middle initial. I also used some names that didn't match the formatting to make sure they did not work.

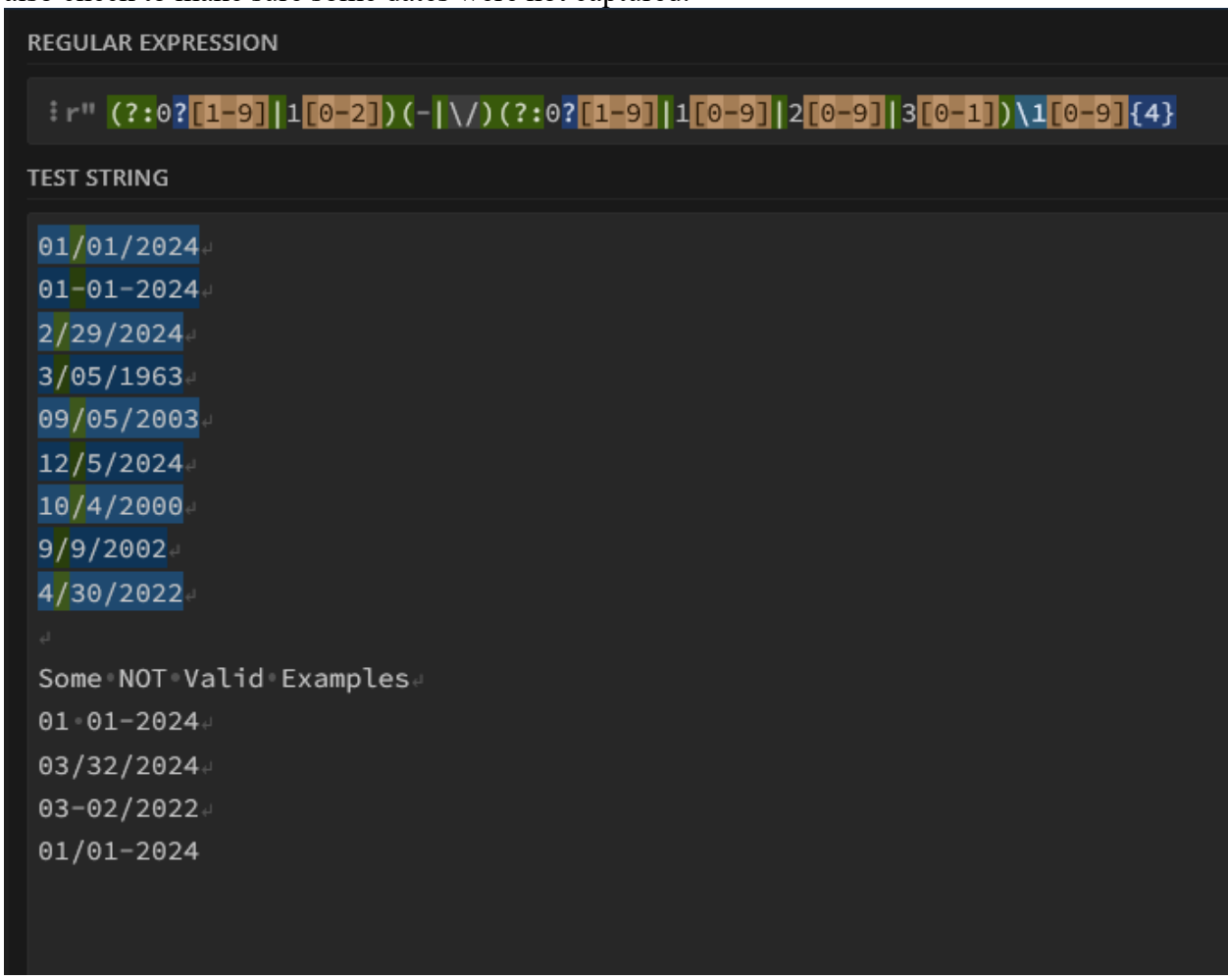


4. Date in MM-DD-YYYY format
   a. (?:0?[1-9]|1[0-2])([-\/])(?:0?[1-9]|1[0-9]|2[0-9]|3[0-1])(?:\1)[0-9]{4}

   b. To create and test this I used regex101.com. I wanted to support an optional 0 in front of the numbers 1-9 in both the month and day, I did this by prefacing the rang 1-9 with 0? That makes the 0 not required but still supported. To support numbers more than 9 but not more than 12 I used and or to see if the number was 1-9 or if it was 1[0-2], saying if the number was 10 through 12 by looking if it starts with a 1 then seeing if the next digit was 0-2 making the number 12. This idea was used in the day segment to go all the way up to 31. In my python file I

wrote a helper method to see if the month is February and checks if it is a leap year. If it is it uses the same expression with the day pattern changed from 0?[1-9]|1[0-9]|2[0-9]|3[0-1]) to 0?[1-9]|1[0-9]|2[0-9]). The only difference between the two is removing the section checking to see if 30+ is in the day. For non leap years the 20 section is set to only go up to 28. For making sure the format was matching and the user did not star with a dash or a / and then switch I captured the first one as a group and used \1 to see if the fist group captured was the same. This enforced the formatting. I then used the range of 0-9 repeated 4 times to enforce a 4 digit date from 0000 to 9999

c. To test this I used a few dates in regex101.com to check for support. I did not test for February 31 because I wrote a helper method in python to check for that. I also check to make sure some dates were not captured.



5. IPv4 address in ###.###.###.###
   a. \b(?:[0-9]|[0-1]?[0-9][0-9]|2[0-4][0-9]|25[0-5])\b\.\b(?:[0-9]|[0-1]?[0-9][0-9]|2[0-4][0-9]|25[0-5])\b\.\b(?:[0-9]|[0-1]?[0-9][0-9]|2[0-4][0-9]|25[0-5])\b(?:\.\b(?:[0-9]|[0-1]?[0-9][0-9]|2[0-4][0-9]|25[0-5])\b)?

b. This regex came out a little bit long but its just the same pattern repeating \b(?:[0-9]|[0-1]?[0-9]?[0-9]|2[0-4][0-9]|25[0-5])\b. Breaking this down \b is word boundary. It forces regex to go though the whole block and make sure that the number isn't spit up into smaller pieces and not read properly. Next is(?: a non-captured group. I didn't want to capture each section of the ip in groups, just the whole IP. The next piece is just checking to see if the number is 1-255. I made the 100 optional to support ip like 33.33.33.33 in the examples provided. I made the last repetition optional to support 3 block IP like in the example doc.

c. For testing I used regex.101 with a bunch of strings I wrote making sure that the IP format is held and no numbers over 255 were allowed