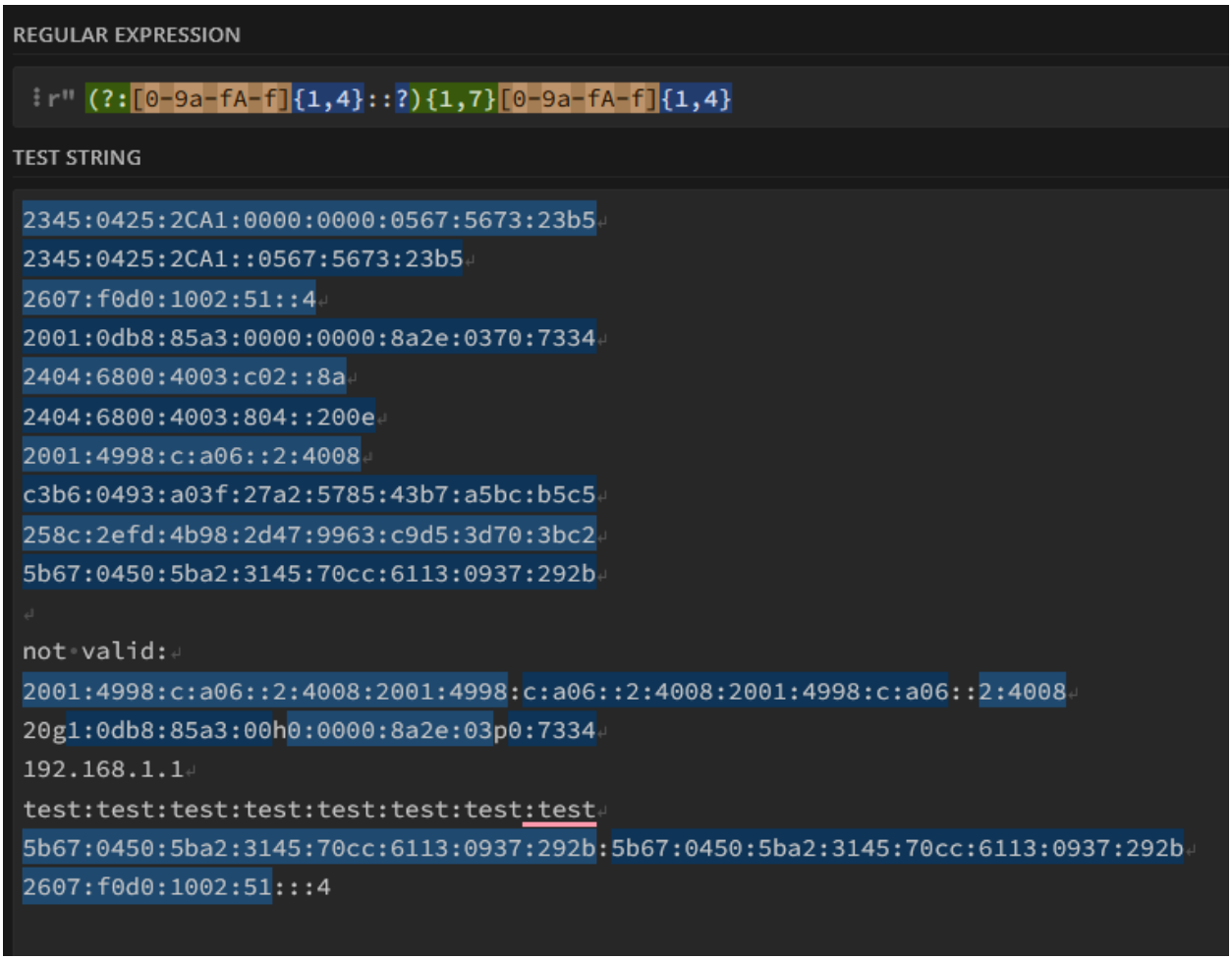
Matthew Adams

CSCD 437 lab6

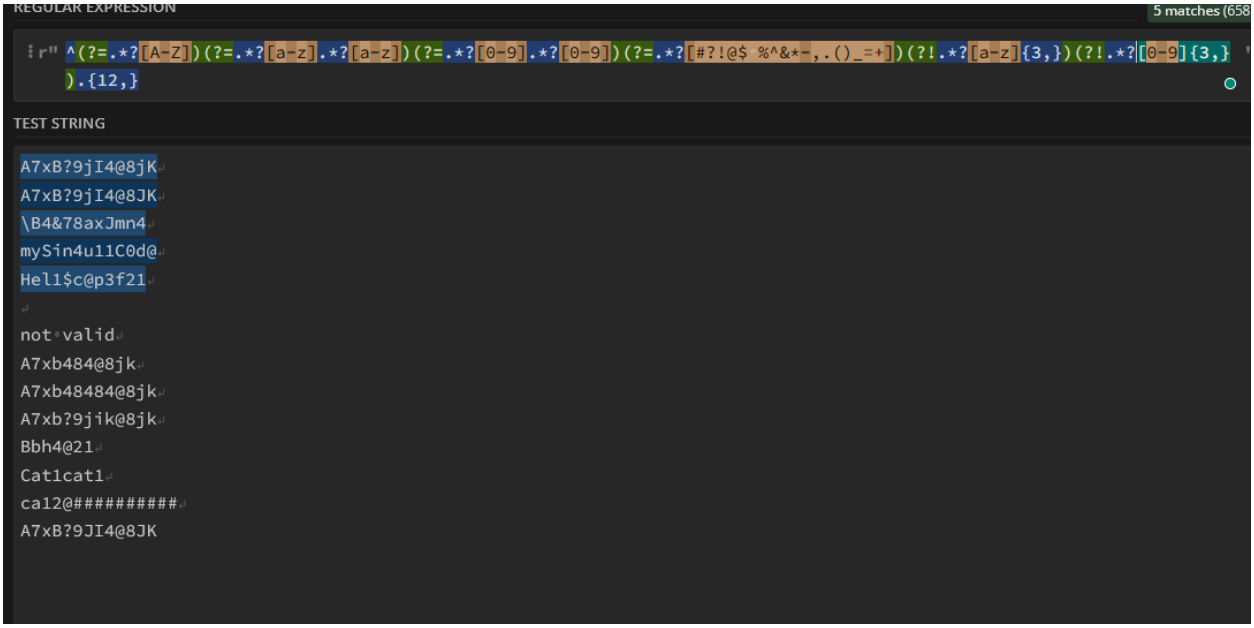
Explanations of regex

1. Ipv6
    a. (?:[0-9a-fA-f]{1,4}::?){1,7}[0-9a-fA-f]{1,4}
        i. This pattern allows for a repeating block of numbers 1-9 and letters up to f. the numbers are repeated up to 4 times per block. I used those specific numbers and letters because ipv6 is hexadecimal which means that the numbers are anything from 0-9 and the letters can any letter from a-f. I also wanted to allow capitals, so I included A- F in the range as well. It is then followed by a colon (:) and another optional colon. I did it this way to allow the :: in ipv6 while still having the ability to catch any blocks behind it.
        ii. For testing I went on to regex101 and used few different online ipv6 generators to test a bunch of valid ip and made up a few that were not valid



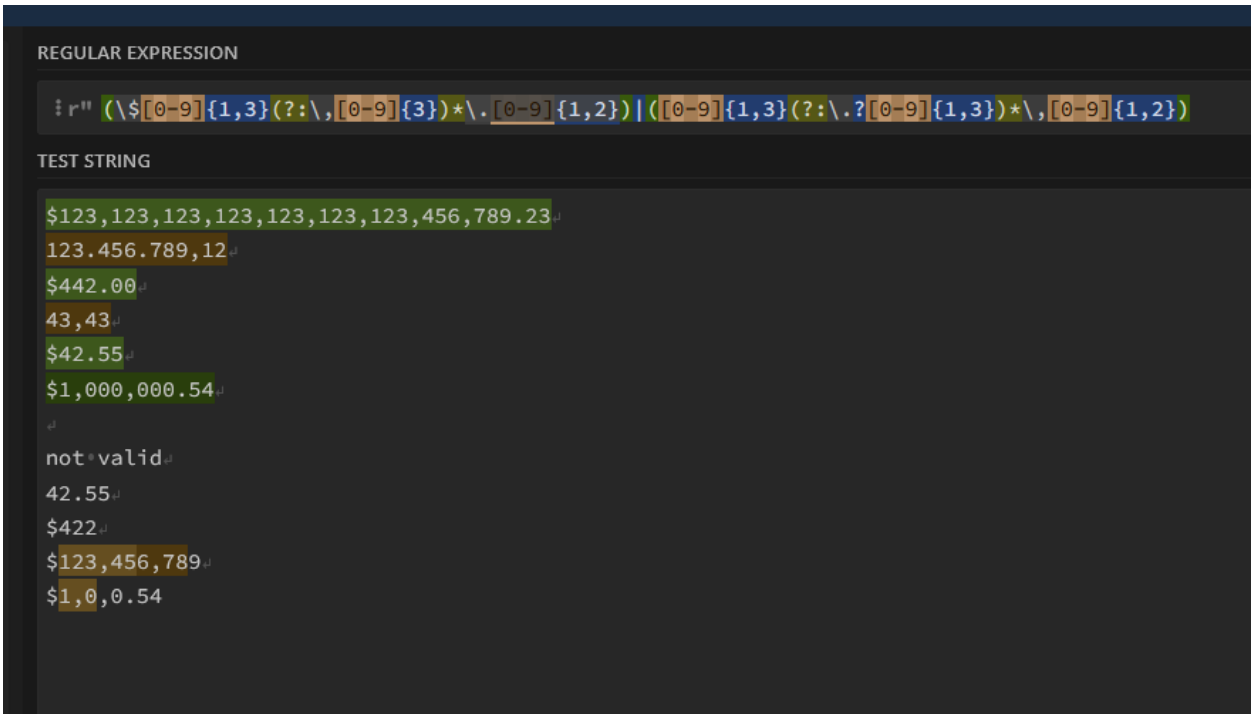
2. Password

a. ^(?=.*?[A-Z])(?=.*?[a-z].*?[a-z])(?=.*?[0-9].*?[0-9])(?=.*?[#?!@$ %^&*-,.()_=+])(?!.*?[a-z]{3,}) (?!.*?[0-9]{3,}). {12,}
   i. For this pattern I started with ^ to make sure I was at the beginning of the string. For the first section (?=.*?[A-Z]) I started with a positive look ahead, followed by any amount of characters, then a range A-Z. this makes sure there is at least one capitol letter by looking through the whole string and finding at least one thing in the range. The next section (?=.*?[a-z].*?[a-z]) is used to look for two lowercase numbers. It begins the same way as the previous section, looking though the entire string for a lowercase letter. However It looks for two lowercase letters, but by repeating .*?[a-z] it is looking for another lowercase letter after it finds the first one. Following this section is (?=.*?[0-9].*?[0-9]) This does the same thing as the previous but with 0-9 rather than a-z. Next is (?=.*?[#?!@$ %^&*-,.()_=+]) this follows the same pattern of looking through the entire string for a character that matches in the rage. In this case the rage is a list of punctuation marks that are allowed in the string. I tried to include as many as I could think of someone using. After these are the negative look ahead, these assert that the following things are not in the string. The first one is (?!.*?[a-z]{3,}) this is making sure that there are not more than three lowercase letters next to each other. It does this with the {3,}, by leaving the second character blank it is like the + saying three or more lowercase letters in a row. The same idea is followed for the next section making sure there are not more than three numbers in a row. After that we make sure that there are at least 12 characters using .{12,}.
   ii. For testing I used regex101.com. I made up a series of passwords that were valid and not valid then checked them.



3. Currency

a. (\$[0-9]{1,3}(?:\,[0-9]{3})*\.[0-9]{1,2})|([0-9]{1,3}(?:\.?[0-9]{1,3})*\,[0-9]{1,2})
   i. For this pattern I made a few assumptions. I required that the penny amount be entered. For this pattern I split it into two using an or statement. The or statement is for us or euro. For the us side the pattern is (\$[0-9]{1,3} (?:\,[0-9]{3})*\.[0-9]{1,2}). The US side has two required parts, \$[0-9]{1,3} and \.[0-9]{1,2}). These two make up the beginning of the us currency and the pennies. The first requires that the dollar sign (\$) and any number 0-9 up two 3 times [0-9]{3}. The other required section makes up the pennies. I only allowed 2 numbers in the penny section because after that it is a dollar [0-9]{2}. I also forced the dot to be required before the penny section \.. The middle section (?:\,[0-9]{3})* is optional and allows it to repeat any number of times, this is done by putting it in a group (I used a non-capturing group (?:) because I did not want to save the middle value separately), and making that group repeat 0 or more times with the *. If it is there it requires the comma then 3 numbers 0-9. The euro section is built the same way but the dots are swapped with commas ([0-9]{1,3}(?:\.?[0-9]{1,3})*\,[0-9]{1,2})
   ii. For testing I used regex.com and wrote some values I deemed to be valid and invalid then wrote the regex around those



4. Email address
   i. ^(?:(?!.*[eE][aA][sS][tT][eE][rR][nN])(?!.*[eE][aA][gG][lL][eE][sS]))(.{2,}(?<!-|\.))@(.{2,})(?<!-|\.)\.(.{2,})(?<!-|\.)
   ii. For this pattern I wanted a lot of flexibility well enforcing the requirements. To start the pattern first looks though the string for eastern or eagle while

ignoring case. It works by checking the letters that spell eastern or eagle show up in that order, and the pattern can pick between the capital or lowercase of that letter. Next we have any character two or more times done with (./{2,}) after that we have a negative lookbehind to make sure that the character before the @ is not a . or -. After that is another block checking for 2+ character and checking to make sure that the next char is not a . before the . the same way it does for the @. This pattern of 2 chars is repeated for the last section of the email.

iii. For testing I used regex101.com. I wrote a few emails that would be valid and not valid and tested for them