Email Address Validator

Documentation

Cohen Beveridge
24/09/2021

Investigation

List of validation rules

1. Total length must be between 6 and 255 characters
2. Address must contain an ‘@’ sign
3. Quoting can be done by enclosing a label in double quotes (")
4. Escaping may be done with a backslash before the character; an escaped backslash does not escape anything else
5. Quotation marks must match unless escaped
6. A quote may either occupy the entire local part, or occupy an entire label, separated by dots, inside the local part
7. An @ sign must occur somewhere after the rightmost quote
8. Quoted labels can include all characters, except for Tab, CR, LF, [,] unless escaped
9. Comments are indicated with parentheses
10. Comments may be within comments, but any opening '(' must have a ')' to match
11. Comments can occur at the start/end of labels or parts
12. Characters within comment must follow the same rules as quoted labels, but square brackets are allowed
13. There must be only one unquoted and uncommented ‘@’ sign
14. The @ sign must have something before it and at least 4 characters after it
15. Both the local part and the domain must not begin with a dot, end with a dot, or contain successive dots unless quoted/commented
16. The local part must be between 1 and 64 characters
17. The local part must not contain these characters:) \ , : ; < > [ ] Tab CR LF unless they are quoted/commented
18. The domain must be a maximum of 253 characters, minimum of 4
19. The domain must contain at least one dot
20. Labels (dot separated strings) must be 63 or less characters
21. The domain must contain only alphabetical characters, digits, dots, and hyphens
22. Labels in the domain must not begin with a hyphen, end with a hyphen, or contain successive hyphens
23. Top level domain (everything after rightmost dot) must be more than one character
24. The Top level domain must only have alphabetical and/or numerical characters
25. Domain can be an ip address enclosed by square brackets “[“, ”]”: either IPv4 or IPv6
26. Comments my occur before or after the square brackets, but not within the brackets
27. IPv4 is 4 integers separated by dots
28. The IPv4 integers must be under 256
29. IPv6 (indicated with “IPv6:”) can be normal format or dual format
30. Normal format is 8 hexadecimal values with 1 – 4 characters
31. Hexadecimal values may have the letters a – f, and digits
32. Dual IPv6 format is 6 hexadecimal values separated by colons “:”, followed by an IPv4 address
33. IPv6 can contain “::” once, representing insignificant zeros (allowing for an undetermined number of parts less than 8/6)

Validation Rules References

http://rumkin.com/software/email/rules.php

https://datatracker.ietf.org/doc/html/rfc2822

https://datatracker.ietf.org/doc/html/rfc3696#section-3

http://haacked.com/archive/2007/08/21/i-knew-how-to-validate-an-email-address-until-i.aspx/

https://www.ietf.org/rfc/rfc5322.html

https://code.iamcal.com/php/rfc822/tests/

http://www.dominicsayers.com/isemail/

https://www.serviceobjects.com/blog/ip-address-vs-domain-in-an-email-address/

These were the most important references I used. A full list is at the end of this document.

Python Research

What is the length of the string?

```
len(string)
```

This function returns the length of the string as an integer

This can be used in the task for detecting if the address exceeds the maximum character count

```
If len(address) > 320:
      Return "invalid"
```

Is a specified character in the string?

```
If character in string:
```

This if statement will run if the specified character is in the string

This can be used in the task for detecting if the address contains an @ symbol

```
If not "@" in address:
      Return "invalid"
```

What character is at a specific location in the string?

```
Character = string[location]
```

This gets the character by slicing the string at the location

This can be used for making sure that the address does not begin with a dot or @ symbol

```
Address[0]
```

What is the location in the string of a specific character?

```
string.find(character)
```

This finds the character in a string and returns the value as an integer. If the character is not in the string, -1 is returned.

This can be used for finding the @ symbol

```
At_symbol = address.find("@")
```

How many of a specific character are in the string?

```
String.count(character)
```

This counts the number of characters in a string and returns the value as an integer.

This can be used to make sure there is only one @ symbol

```
Address.count("@")
```

How can a string be split into smaller sub-strings?

```
Split()
```

This splits a string into two parts and stores the new strings in a list

```
parts = address.split("@")
localpart = parts[0]
domain = parts[1]
```

References:

https://www.w3schools.com/python/python_strings.asp

Design

PSUEDOCODE

Rules 1 & 2: Checks Length of Email Address

```
MODULE LengthInRange (address)
BEGIN
        Length ← length of address
        IF length > 256 OR length < 6
                THEN return False
                ELSE return True
        END IF
END
```

Rule 3: Checks if address contains @

```
MODULE AddressContainsAt (address)
BEGIN
        IF '@' not in address
                THEN return ← False
                ELSE return ← True
        END IF
END
```

MODULE to return address with empty quotes, so as not to interfere with other validation rules

```
MODULE EmptyQuotes (OriginalAddress)
BEGIN
    address ← OriginalAddress
    WHILE ‘ ” ’ in address
        OpenQuote ← find(0, address, ‘ “ ’)
        WHILE address[OpenQuote – 1] = ‘\’
            OpenQuote ← find(0, address, ‘ “ ’)
        End While
        CloseQuote ← find(OpenQuote, address, ‘ “ ’)
        WHILE address[CloseQuote – 1] = ‘\’
            CloseQuote ← find(OpenQuote, address, ‘ “ ’)
        End While
        BeforeQuote ← SubString(0, OpenQuote – 1, address)
        AfterQuote ← SubString(CloseQuote + 1, len(address), address)
        Address ← BeforeQuote + AfterQuote
        OutputAddress ← OutputAddress + BeforeQuote + ‘ "" ’ + AfterQuote
                                                    ## (includes quote marks but not contents)
    End While
    RETURN OutputAddress
END
```

Rule 5: Find rightmost quote (if any) and check for an @ further right

```
MODULE AtAfterQuotes (address)
BEGIN
        LastQuote ← find(0, address, ‘ “ ’)
        AfterQuotes ← SubString(LastQuote, len(address), address)
        WHILE ‘ “ ’ in AfterQuotes
                LastQuote ← find(0, address, ‘ “ ’)
                AfterQuotes ← SubString(LastQuote, len(address), address)
        End While
        IF ‘@’ in AfterQuotes
                THEN RETURN True
                ELSE RETURN False
        End If
END
```

Rule 6: Validate Contents of Quotes

```
MODULE QuoteContentsValid (OriginalAddress)
BEGIN
    Address ← OriginalAddress
    QuoteList ← []
    WHILE ‘ ” ’ in address
        OpenQuote ← find(0, address, ‘ “ ’)
        WHILE address[OpenQuote – 1] = ‘\’
            OpenQuote ← find(0, address, ‘ “ ’)
        End While
        CloseQuote ← find(OpenQuote, address, ‘ “ ’)
        WHILE address[CloseQuote – 1] = ‘\’
            CloseQuote ← find(OpenQuote, address, ‘ “ ’)
        End While
        Quote ← SubString(OpenQuote + 1, CloseQuote – 1, address)
        Append Quote to QuoteList
        BeforeQuote ← SubString(0, OpenQuote – 1, address)
        AfterQuote ← SubString(CloseQuote + 1, len(address), address)
        Address ← BeforeQuote + AfterQuote
    End While
    Restricted ← ‘Tab’, ‘CR’, ‘LF’, ‘[’, ‘]’
    FOR Quote in QuoteList
        FOR Character in range(len(Quote))
            IF Quote[Character] in Restricted AND Quote[Character – 1] != ‘\’
                THEN RETURN False
            End If
        End For
    End For
    RETURN True
END
```

Rule 7: Validate Position of Quotes

```
MODULE QuotePositionValid (OriginalAddress)
BEGIN
        address ← OriginalAddress
        WHILE ‘ ” ’ in address
                OpenQuote ← find(0, address, ‘ “ ’)
                WHILE address[OpenQuote – 1] = ‘\’
                        OpenQuote ← find(0, address, ‘ “ ’)
                End While
                CloseQuote ← find(OpenQuote, address, ‘ “ ’)
                WHILE address[CloseQuote – 1] = ‘\’
                        CloseQuote ← find(OpenQuote, address, ‘ “ ’)
                End While
                IF OpenQuote != 0 AND address[OpenQuote – 1] != ‘.’ AND
address[CloseQuote + 1] != ‘@’ AND address[CloseQuote + 1] != ‘.’
                        THEN RETURN False
                End If
        End While
        RETURN True
END
```

MODULE to return address without comments, so as not to interfere with other validation rules

```
MODULE RemoveComments (OriginalAddress)
BEGIN
    address ← CALL: EmptyQuotes (OriginalAddress)
    WHILE ‘ ( ’ in address
        OpenBracket ← find(0, address, ‘ ( ’)
        CloseBracket ← find(0, address, ‘ ) ’)
        WHILE address[CloseBracket – 1] = ‘\’
            CloseBracket ← find(CloseBracket, address, ‘ ) ’)
        End While
        BeforeComment ← SubString(0, OpenBracket – 1, address)
        AfterComment ← SubString(CloseBracket + 1, len(address), address)
        Address ← BeforeComment + AfterComment
    End While
    RETURN address
END
```

Rule 9: Making sure commented comments match up, and no extra brackets exist, within comments

```
MODULE CommentedComments (OriginalAddress)
BEGIN
    CommentList ← []
    address ← CALL: EmptyQuotes (OriginalAddress)
    WHILE ‘ ( ’ in address
        OpenBracket ← find(0, address, ‘ ( ’)
        CloseBracket ← find(0, address, ‘ ) ’)
        WHILE address[CloseBracket – 1] = ‘\’
            CloseBracket ← find(CloseBracket, address, ‘ ) ’)
        End While
        Comment ← SubString(OpenBracket, CloseBracket, address)
        Append Comment to CommentList
        BeforeComment ← SubString(0, OpenBracket – 1, address)
        AfterComment ← SubString(CloseBracket + 1, len(address), address)
        Address ← BeforeComment + AfterComment
    End While
    FOR Comment in CommentList
        OpenCount ← count( ‘(’, Comment)
        CloseCount ← count( ‘)’, Comment)
        IF OpenCount != CloseCount          ## For comments to be right, brackets must match ##
            THEN RETURN False
        End If
    End For
END
```

Rule 10: Validate Position of Comments

```
MODULE CommentPositionValid (OriginalAddress)
BEGIN
        address ← CALL: EmptyQuotes (OriginalAddress)
        WHILE ‘ ( ’ in address
                OpenBracket ← find(0, address, ‘ ( ’)
                CloseBracket ← find(0, address, ‘ ) ’)
                WHILE address[CloseBracket – 1] = ‘\’
                        CloseBracket ← find(CloseBracket, address, ‘ ) ’)
                End While
                IF OpenBracket != 0 AND address[OpenBracket – 1] != ‘.’ AND
address[OpenBracket – 1] != ‘@’ AND address[CloseBracket + 1] != ‘.’ AND
address[CloseBracket + 1] != ‘@’
                        THEN RETURN False
                End If
        End While
        RETURN True
END
```

Rule 11: Validate Contents of Comments

```
MODULE CommentContentsValid (OriginalAddress)
BEGIN
    Address ← CALL: EmptyQuotes (OriginalAddress)
    CommentList ← []
    WHILE ‘ ( ’ in address
        OpenBracket ← find(0, address, ‘ ( ’)
        CloseBracket ← find(0, address, ‘ ) ’)
        WHILE address[CloseBracket – 1] = ‘\’
            CloseBracket ← find(CloseBracket, address, ‘ ) ’)
        End While
        Comment ← SubString(OpenBracket + 1, CloseBracket – 1, address)
        Append Comment to CommentList
        BeforeComment ← SubString(0, OpenBracket – 1, address)
        AfterComment ← SubString(CloseBracket + 1, len(address), address)
        Address ← BeforeComment + AfterComment
    End While
    Restricted ← ‘Tab’, ‘CR’, ‘LF’, ‘[’, ‘]’
    FOR Comment in CommentList
        FOR Char in range(len(Comment))
            IF Comment[Char] in Restricted AND Comment[Char – 1] != ‘\’
                THEN RETURN False
            End If
        End For
    End For
    RETURN True
END
```

Rule 12: Only one unquoted & uncommented @

```
MODULE OnlyOneAt (OriginalAddress)
BEGIN
        Address ← CALL: EmptyQuotes (OriginalAddress)
        Address ← CALL: RemoveComments (address)
        AtNum ← count('@', address)
        IF AtNum = 1
                THEN RETURN True
                ELSE RETURN False
        End If
END
```

Rule 13: Validate Position of @

```
MODULE ValidateAtPos (OriginalAddress)
BEGIN
        Address ← CALL: EmptyQuotes (OriginalAddress)
        Address ← CALL: RemoveComments (address)
        AtPos ← find(0, '@', address)
        IF (AtPos > 1) AND (AtPos < (len(address)) – 3))
                THEN RETURN True
                ELSE RETURN False
        End If
END
```

MODULE to find and return the position of the true @

```
MODULE FindAt (OriginalAddress)
BEGIN
    Address ← CALL: EmptyQuotes (OriginalAddress)
    Address ← CALL: RemoveComments (address)
    AtPos ← find(0, '@', address)
    RETURN AtPos
END
```

MODULE to find and return local part

```
MODULE FindLocal (address)
BEGIN
    AtPos ← CALL: FindAt (address)
    LocalPart ← left(address, AtPos)
    RETURN LocalPart
END
```

MODULE to find and return domain

```
MODULE FindDomain (address)
BEGIN
    AtPos ← CALL: FindAt (address)
    Domain ← right(address, AtPos)
    RETURN Domain
END
```

Rule 14: Check if Local Part is a valid length

```
MODULE LocalLength (address)
BEGIN
        LocalPart ← CALL: FindLocal (address)
        LocalLength ← len(LocalPart)
        IF (LocalLength > 0) AND (LocalLength < 65)
                THEN RETURN True
                ELSE RETURN False
        End If
END
```

Rule 15: Check if Domain is a valid length

```
MODULE DomainLength (address)
BEGIN
        Domain ← CALL: FindDomain (address)
        DomainLength ← len(Domain)
        IF (DomainLength > 3) AND (DomainLength < 254)
                THEN RETURN True
                ELSE RETURN False
        End If
END
```

Rule 16: Find Labels and check if they are valid lengths

```
MODULE LabelLength (OriginalAddress)
BEGIN
    Address ← CALL: EmptyQuotes (OriginalAddress)
    Address ← CALL: RemoveComments (address)
    DotLocations ← []
    OneDot ← find(0, '.', address)
    FOR Dot in range(count('.', address))
        Append OneDot to DotLocations
        OneDot ← find(OneDot, '.', address)
    End For
    Labels ← []
    LabelStart ← 0
    LabelEnd ← DotLocations[0]
    FOR Dot in range(length(DotLocations) – 1)
        Label ← SubString(Dot, Dot + 1, address)
        Append Label to Labels
    End For
    FOR Label in Labels
        IF len(Label) > 63
            RETURN False
        End if
    End For
END
```

Rule 17: Checks that domain contains at least one dot

```
MODULE DotInDomain (address)
BEGIN
    Domain ← CALL: FindDomain (address)
    IF '.' In Domain
        THEN RETURN True
        ELSE RETURN False
    End If
END
```

MODULE to find Top Level Domain

```
MODULE FindTLD (address)
BEGIN
    LastDot ← 0
    FOR char IN range(len(address))
        IF address[char] = '.'
            THEN LastDot ← char
        End If
    End For Loop
    TLD ← SubString(LastDot, len(address), address)
    RETURN TLD
END
```

Rule 18: Checks Top Level Domain is more than one character

```
MODULE TLD_MoreThanOne (address)
BEGIN
    TLD ← CALL: FindTLD (address)
    IF '.' In TLD
        THEN RETURN True
        ELSE RETURN False
    End If
END
```

Rule 19: Checks for dot invalidity

```
MODULE DotsCorrect (OriginalAddress)
BEGIN
    Address ← CALL: EmptyQuotes (OriginalAddress)
    Address ← CALL: RemoveComments (address)
    IF (address[0] = '.') OR address[len(address)] = '.'
        THEN RETURN False
    ELSE
        FOR char in range(len(address))
            IF address[char] = '.'
            THEN
                IF (address[char – 1] = '.') OR (address[char + 1] = '.')
                    THEN RETURN False
                End If
            End If
        End For Loop
        RETURN True
    End If
END
```

Rule 20: Checks for invalid characters in local part

```
MODULE LocalCharsCorrect (address)
BEGIN
        LocalPart ← CALL: FindLocal (address)
        Restricted ← " ,:;<>[]"
        FOR char in LocalPart
                IF char in Restricted
                        THEN RETURN False
                End IF
        End For Loop
        RETURN True
END
```

Rule 21: Checks for invalid characters on domain

```
MODULE DomainCharsCorrect (address)
BEGIN
        Domain ← CALL: FindDomain (address)
        Allowed ← "-1234567890abcdefghijklmnopqrstuvwxyz"
        FOR char in Domain
                IF char not in Allowed
                        THEN RETURN False
                End If
        End For loop
END
```

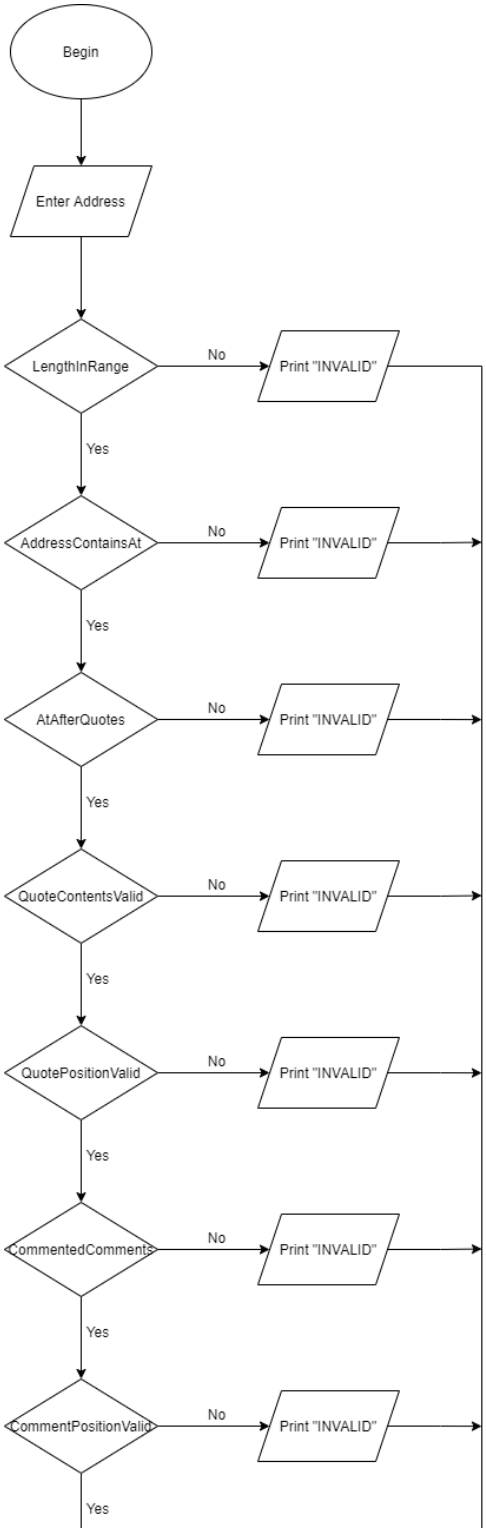
Rule 22: Checks for Hyphen validity

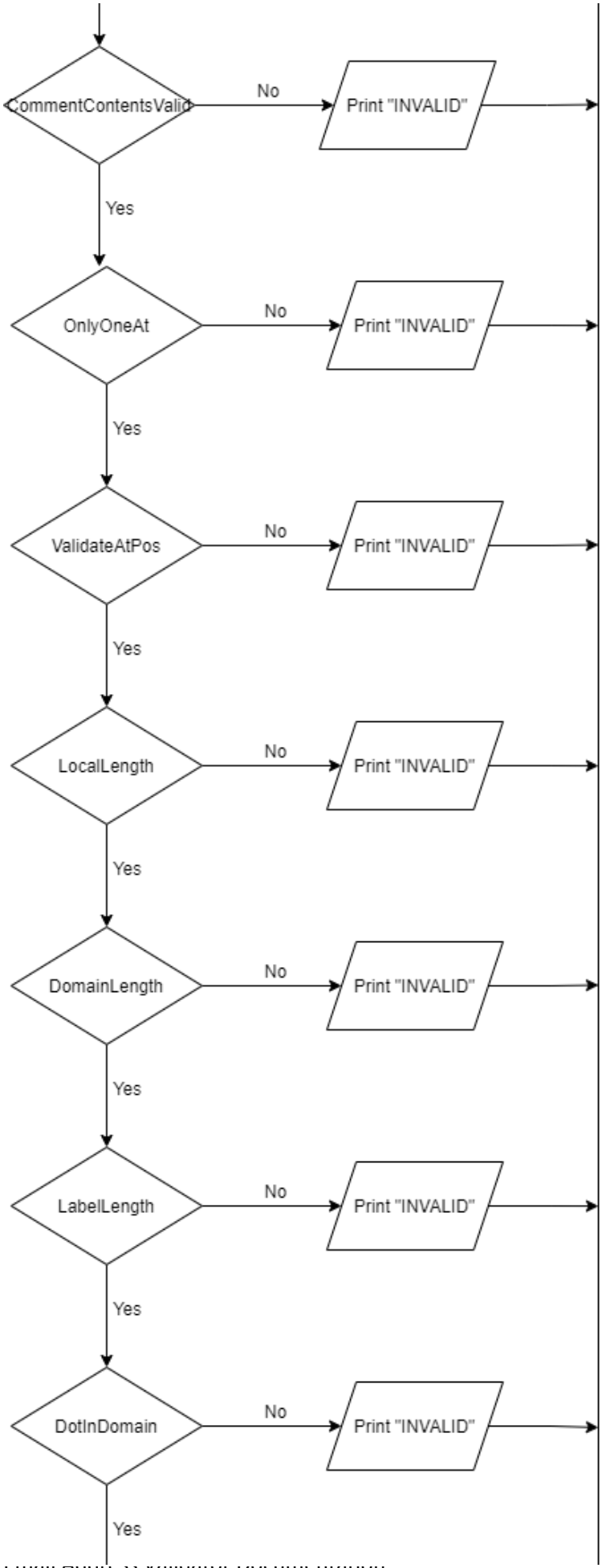
```
MODULE HyphensCorrect (address)
BEGIN
    Domain ← CALL: FindDomain (address)
    IF (Domain[0] = '-') OR Domain[len(address)] = '-'
        THEN RETURN False
    ELSE
        FOR char in range(len(Domain))
            IF Domain[char] = '-'
            THEN
                IF (Domain[char – 1] = '-') OR (Domain[char + 1] = '-')
                    THEN RETURN False
                End If
            End If
        End For Loop
        RETURN True
    End If
END
```

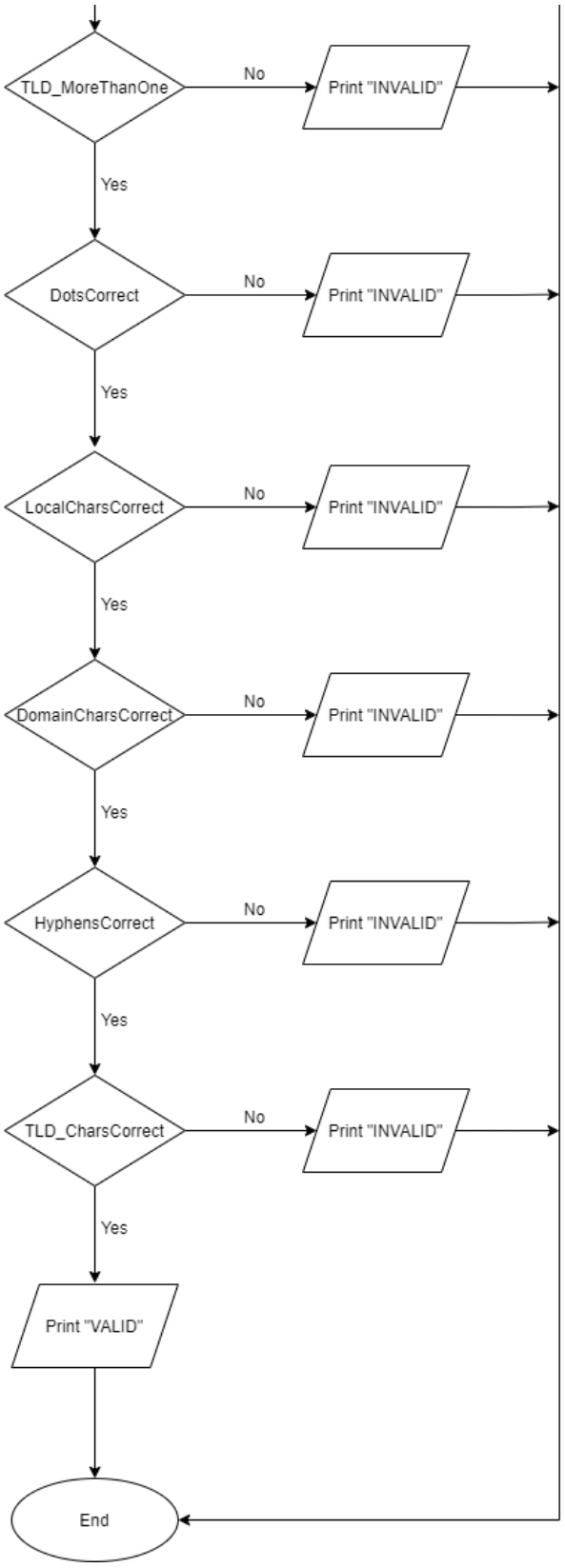
Rule 23: Checks for Top Level Domain character validity

```
MODULE TLD_CharsCorrect (address)
BEGIN
    TLD ← CALL: FindTLD (address)
    Alpha ← "abcdefghijklmnopqrstuvwxyz"
    FOR char in TLD
        IF char not in Alpha
            THEN RETURN False
        End If
    End For
END
```

Mainline Flowchart:




CommentContentsValid
No
Print "INVALID"
Yes
OnlyOneAt
No
Print "INVALID"
Yes
ValidateAtPos
No
Print "INVALID"
Yes
LocalLength
No
Print "INVALID"
Yes
DomainLength
No
Print "INVALID"
Yes
LabelLength
No
Print "INVALID"
Yes
DotInDomain
No
Print "INVALID"
Yes



** Each module enters the same parameter (email) and returns either True or False

Evaluation

Interactive mode

To test my interactive mode program, I inputted different types of valid and invalid email addresses. At first, many did not output the correct result, but after debugging, most are correct. Some examples of valid and invalid email addresses are listed. I used these when testing my interactive mode.

Basic valid addresses:

test@test.com, a@a.io, 123@123.123, TEST@test.COM, test@test.test.com, test@test1.2com, test@test-test.com, !#$%^&*@test.com

Complex valid addresses:

"test"@test.com, "test\"test"@test.com, "test\\"@test.com, "test"."test"@test.com, "test".test."test"@test.com, "@@@"@test.com, "!E#H@#(H#@"@test.com, (test)test(test).(test)test(test)@(test)test(test).(test)com(test), (t(e(s(t)s)e)t)test@t.com, test@(te\(st)test.com, test@test.com(@@@)

test@[1.1.1.1], test@[IPv6:1:1:1:1:1:1:1:1], test@[IPv6:a:a:a:a:a:a:a:a], test@[IPv6:a:a:a:a:a:a:1.1.1.1]

Basic invalid addresses:

Test, test@@test.com, test@com, a@test@test.com, test@###.com, test@te--st.com, test@t.c, test..test@test.com, .test@test.com, test.@test.com, test@-test.com, test@test.com-, te;;;st@test.com, te\st@test.com

Complex invalid addresses:

"te"st"@test.com, aa"test"aa@test.com, test"test"@test.com, "test\"@test.com, aa(bb)aa@test.com, test@test.com(a(a(a(a(a)b)b)b)b)

test@[a.a.a.a], test@[1.1.1], test@[1.1.1.257], test@[IPv7:1:1:1:1:1:1:1:1], test@[IPv6:z:z:z:z:z:z:z:z:z]

I used test addresses like these and more to test my interactive mode program. How I did it was I inputted the address into the program, and I could see if the result was the one I expected, or if it was not. A method I used for testing specific validating functions was copying them into a test python file so that I could test the function individually and simply, and I could narrow down any problem which I might have had.

Of the basic addresses, they were mostly fine from the first time I tested them. The complex addresses have quotes, comments, and IP addresses. These were a little harder to get right. I had to

use these test addresses to troubleshoot their functions, because most of them returned an incorrect result after the very first test.

One problem was typing the test addresses into the program every time I made a changed which affected it. What I should have done was to make a list in the actual python code, and gotten the code to run off that, just for the testing. That would have made troubleshooting more efficient.

Batch Mode

After I believed that I had finished the functions, I created a Batch mode of the program which had file-reading capabilities. When I inputted test files into the program and checked the results in the new file, I found that my results had some differences to the answers files. With this knowledge, I edited some functions which should have dealt with the errors, and eventually my output files became very similar to the answer files. However, some answers to email addresses in the files did not match my research, so I decided to leave them as they were. I have given a test file (testfile.txt) which contains a collection of test addresses from resources which were given.

Summary Evaluation

I think that the investigation phase was the most difficult in this task. There were many different resources to use, and a lot of them contradicted each other. Because of this, it was quite difficult to determine which source was correct, and the sources I chose might not have actually been right in every aspect. My list of validation rules is very different now compared to the beginning of the build phase because I was often finding new rules which I hadn't previously considered.

The design phase was challenging for me because I had to think about what the program should do exactly, without actually testing it as I went along. My pseudocode looks very different to my now finished python code. The mainline flowchart which I made was very simple, and I did not consider interactive/batch modes or specific error messages when I was creating it.

The build phase took the longest out of all the phases because it was difficult to get everything right. The basic functions (like length and @ symbol) didn't take too long, but the complex functions took a lot longer. These were the functions which dealt with quotes, comments, and IP addresses. One of the biggest problems I had was when quotes and comments were combined. Because a quote inside a comment is not a quote, and a comment in a quote is not a comment. My original plan was to deal with them separately, but I realised I had to do them simultaneously. This was quote challenging, but I managed to get it to work. IP address were the other major challenge, but a lot of research and

experimentation allowed it to work. Although the build phase was somewhat difficult, I did enjoy it a lot. It was by far my favourite part of the process.

The test phase was helpful for me, as it allowed me to pinpoint the errors in my code and straighten them out. Creating batch mode was something which I thought would be tough, but it turned out to actually be quite simple. The batch mode really helped with testing, as it allowed me to test large lists of email addresses all at once.

Overall, I think this was a good task. I did not like it at first, during the investigation and research phase, but I did enjoy the building phase of the task. I am glad that I was able to finish it with time to spare. If I was to do this task again, I wouldn't change much, except maybe focussing more on the investigation phase at the beginning instead of still during research all through the task. I think I have learnt a few new Python skills during this task, and I think I did a good job on it.

By Cohen Beveridge

References

I used many references during the course of this task. Here they are, listed.

Akins, T. (n.d.). Email Validation Done Right. http://rumkin.com/software/email/

Clarke, B. (2021). eMail Validator Activity Outline. eMail Validator Activity Outline.pdf

Comparing E-mail Address Validating Regular Expressions (2012). CN Blogs. https://www.cnblogs.com/hyqing/p/3421730.html

Fuentes, E. (2019). IP address vs domain in an email address. https://www.serviceobjects.com/blog/ip-address-vs-domain-in-an-email-address/

Haack, P. (2007). I Knew How To Validate An Email Address Until I Read The RFC. http://haacked.com/archive/2007/08/21/i-knew-how-to-validate-an-email-address-until-i.aspx/

Henderson, C. (n.d.). RFC 822 Email Address Parser in PHP. https://code.iamcal.com/php/rfc822/

Hinden, R. (2006). RFC 4291 - IP Version 6 Addressing Architecture. https://datatracker.ietf.org/doc/html/rfc4291

JavaScript form validation - checking email (n.d.). W3 resources. https://www.w3resource.com/javascript/form/example-javascript-form-validation-email-REC-2822.html

Klensin, J. (2004). RFC 3696 - Application Techniques for Checking and Transformation of Names. https://datatracker.ietf.org/doc/html/rfc3696

Nagar, R. (n.d.). JMail Email Address Validation. https://www.rohannagar.com/jmail/

Resnick, P. (2008). RFC 5322 - Internet Message Format. https://datatracker.ietf.org/doc/html/rfc5322

Wikipedia. (2021). Email address. https://en.wikipedia.org/wiki/Email_address