BCA – 502: Python Programming

Rahul Kumar Singh

In today's Class we have discussed on Password, email, url validation using regular expression in Python.

Regular Expression Patterns:-

Except for control characters, (+ ? . * ^ $ ( ) [ ] { } | \), all characters match themselves. You can escape a control character by preceding it with a backslash.

Following table lists the regular expression syntax that is available in Python –

^ - Matches beginning of line.

$ - Matches end of line.

. - Matches any single character except newline. Using m option allows it to match newline as well.

[...] - Matches any single character in brackets.

[^...] - Matches any single character not in brackets

re* - Matches 0 or more occurrences of preceding expression.

re+ - Matches 1 or more occurrence of preceding expression.

re? - Matches 0 or 1 occurrence of preceding expression.

re{n} - Matches exactly n number of occurrences of preceding expression.

re{n,} - Matches n or more occurrences of preceding expression.

re{n, m} - Matches at least n and at most m occurrences of preceding expression.

a| b - Matches either a or b.

(re) - Groups regular expressions and remembers matched text.

(?imx) - Temporarily toggles on i, m, or x options within a regular expression. If in parentheses, only that area is affected.

(?-imx) - Temporarily toggles off i, m, or x options within a regular expression. If in parentheses, only that area is affected.

(?:re) - Groups regular expressions without remembering matched text.

(?imx:re) - Temporarily toggles on i, m, or x options within parentheses.

(?-imx:re) - Temporarily toggles off i, m, or x options within parentheses.

(?#...) - Comment.

(?=re) - Specifies position using a pattern. Doesn't have a range.

(?!re) - Specifies position using pattern negation. Doesn't have a range.

(?> re) - Matches independent pattern without backtracking.

\w - Matches word characters.

\W - Matches nonword characters.

\s - Matches whitespace. Equivalent to [\t\n\r\f].

\S - Matches nonwhitespace.

\d - Matches digits. Equivalent to [0-9].

\D - Matches nondigits.

\A - Matches beginning of string.

\Z - Matches end of string. If a newline exists, it matches just before newline.

\z - Matches end of string.

\G - Matches point where last match finished.

\b - Matches word boundaries when outside brackets. Matches backspace (0x08) when inside brackets.

\B - Matches nonword boundaries.

\n, \t - Matches newlines, carriage returns, tabs.

\1...\9 - Matches nth grouped subexpression.

\10 - Matches nth grouped subexpression if it matched already. Otherwise refers to the octal representation of a character code.

[Pp]ython - Match "Python" or "python"

rub[ye] - Match "ruby" or "rube"

[aeiou] - Match any one lowercase vowel

[0-9] - Match any digit; same as [0123456789]

[a-z] - Match any lowercase ASCII letter

[A-Z] - Match any uppercase ASCII letter

[a-zA-Z0-9] - Match any of the above

[^aeiou] - Match anything other than a lowercase vowel

[^0-9] - Match anything other than a digit

. - Match any character except newline

\d - Match a digit: [0-9]

\D - Match a nondigit: [^0-9]

\s - Match a whitespace character: [\t\r\n\f]

\S - Match nonwhitespace: [^ \t\r\n\f]

\w - Match a single word character: [A-Za-z0-9_]

\W - Match a nonword character: [^A-Za-z0-9_]

(['"])[^\1]*\1 - Single or double-quoted string. \1 matches whatever the 1st group matched. \2 matches whatever the 2nd group matched, etc.

\d{3} - Match exactly 3 digits

\d{3,} - Match 3 or more digits

\d{3,5} - Match 3, 4, or 5 digits

<.*> - Greedy repetition: matches "<python>perl>"

<.*?> - Nongreedy: matches "<python>" in "<python>perl>"

Password validation in Python using regular expression:-

Take a password as a combination of alphanumeric characters along with special characters, and check whether the password is valid or not with the help of few conditions.

Conditions for a valid password are:

- Should have at least one number.
- Should have at least one uppercase and one lowercase character.
- Should have at least one special symbol.
- Should be between 6 to 20 characters long.

Example:-

```
# importing re library

import re

def main():

    passwd = 'Lnt@123'
```

```
    reg  =  "^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*#?&])[A-
Za-z\d@$!#%*?&]{6,20}$"

    # compiling regex

    pat = re.compile(reg)

    # searching regex

    mat = re.search(pat, passwd)

    # validating conditions

    if mat:

        print("Password is valid.")

    else:

        print("Password invalid !!")

# Driver Code

if __name__ == '__main__':

    main()
```

Output:

Password is valid.

Url validation in Python using regular expression:-

Here the idea is to use Regular Expression to validate a URL.

- ➤ Get the URL.

- Create a regular expression to check the valid URL as mentioned below:

  regex = “((http|https)://)(www.)?”

+ “[a-zA-Z0-9@:%._\\+~#?&//=]{2,256}\\.[a-z]”

+ “{2,6}\\b([-a-zA-Z0-9@:%._\\+~#?&//=]*)”

- The URL must start with either http or https and then followed by :// and then it must contain www. and then followed by subdomain of length (2, 256) and last part contains top level domain like .com, .org etc.
- Match the given URL with the regular expression.
- Return true if the URL matches with the given regular expression, else return false.

Example:-

```
import re

def isValidURL(str):

    # Regex to check valid URL

    regex = ("((http|https)://)(www.)?" +

             "[a-zA-Z0-9@:%._\\+~#?&//=]" +

             "{2,256}\\.[a-z]" +

             "{2,6}\\b([-a-zA-Z0-9@:%" +

             "._\\+~#?&//=]*)")
```

```
    p = re.compile(regex)     # Compile the ReGex
    if (str == None):         # If the string is empty return false
        return False
    if(re.search(p, str)):    # Return true if the string match
        return True
    else:
        return False
# Driver code Test Case 1:
url = "https://www.lntcollege.ac.in/"
if(isValidURL(url) == True):
    print("Yes")
else:
    print("No")
```

Output:-

Yes

Email validation in Python using regular expression:-

An email is a string (a subset of ASCII characters) separated into two parts by @ symbol, a “personal_info” and a domain, that is personal_info@domain.

Example:-

```
import re
# Define a function for
# for validating an Email
def check(s):
    pat = r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b'
    if re.match(pat,s):
        print("Valid Email")
    else:
        print("Invalid Email")
# Driver Code
if __name__ == '__main__':
    # Enter the email
    email = "ankitraj326@gmail.com"
    # calling run function
    check(email)
    email = "lnt.muz@our-college.org"
    check(email)
    email = "ankitraj326.com"
    check(email)
```

Output:-

Valid Email

Valid Email

Invalid Email