K. J. Somaiya College of Engineering, Mumbai-77

Department of Computer Engineering

Roll No.: 1911027

Experiment No. 7

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of the Staff In-charge with

Problem Statement:
Accept the IP address as input and Identify the validity of the IP address using regular expression and display the class of IP Address.

AIM: Use concepts of Python's regular expressions

Expected OUTCOME of Experiment:

CO1: Describe the Numbers, Math functions, Strings, List, Tuples and Dictionaries in Python.

CO2: Interpret different Decision Making statements, Functions, Object oriented programming in Python.

Books/ Journals/ Websites referred:

1) https://www.tutorialspoint.com/python/python_reg_expressions
2) https://www.w3schools.com/python/python_regex.asp
3) https://docs.python.org/3/howto/regex.html
4) https://www.geeksforgeeks.org/regular-expression-python-examples-set-1/

Pre Lab/ Prior Concepts:

Regular Expressions:

A regular expression is a special sequence of characters that helps you match or find other strings or sets of strings, using a specialized syntax held in a pattern. Regular expressions are widely used in UNIX world. Python has a built-in package called re, which can be used to work with Regular Expressions. The Python module re provides full support for Perl-like regular expressions in Python. The re module raises the exception re.error if an error occurs while compiling or using a regular expression. There are various characters, which would have special meaning when they are used in regular expression. To avoid any confusion while dealing with regular expressions, we would use Raw Strings as r'expression'. Regular expression patterns are compiled into a series of bytecodes which are then executed by a matching engine written in C. For advanced use, it may be necessary to pay careful attention to how the engine will execute a given RE, and write the RE in a certain way in order to produce bytecode that runs faster.

The match Function:

This function attempts to match RE pattern to string with optional flags. Here is the syntax for this function −

re.match(pattern, string, flags=0)

Here is the description of the parameters −

Sr.No.	Parameter & Description
1	**pattern** This is the regular expression to be matched.
2	**string** This is the string, which would be searched to match the pattern at the beginning of string.
3	**flags** You can specify different flags using bitwise OR (|). These are modifiers, which are listed in the table below.

The re.match function returns a match object on success, None on failure. We usegroup(num) or groups() function of match object to get matched expression.

Sr.No.	Match Object Method & Description
1	**group(num=0)** This method returns entire match (or specific subgroup num)
2	**groups()** This method returns all matching subgroups in a tuple (empty if there weren't any)

Example:

```
import re

line = "Cats are smarter than dogs"

matchObj = re.match( r'(.*) are (.*?) .*', line, re.M|re.I)

if matchObj:

   print "matchObj.group() : ", matchObj.group()
```

```
   print "matchObj.group(1) : ", matchObj.group(1)

   print "matchObj.group(2) : ", matchObj.group(2)

else:

   print "No match!!"
```

When the above code is executed, it produces following result –

matchObj.group() : Cats are smarter than dogs

matchObj.group(1) : Cats

matchObj.group(2) : smarter

The search Function:

This function searches for first occurrence of RE pattern within string with optional flags. Here is the syntax for this function –

re.search(pattern, string, flags=0)

Here is the description of the parameters –

Sr.No.	Parameter & Description
1	**pattern** This is the regular expression to be matched.
2	**string** This is the string, which would be searched to match the pattern anywhere in the string.
3	**flags** You can specify different flags using bitwise OR (|). These are modifiers, which are listed in the table below.

The re.search function returns a match object on success, none on failure. We use group(num) or groups() function of match object to get matched expression.

Sr.No.	Match Object Methods & Description
1	**group(num=0)** This method returns entire match (or specific subgroup num)
2	**groups()** This method returns all matching subgroups in a tuple (empty if there weren't any)

Example:

```
import re

line = "Cats are smarter than dogs";

searchObj = re.search( r'(.*) are (.*?) .*', line, re.M|re.I)

if searchObj:
```

```
   print "searchObj.group() : ", searchObj.group()
   print "searchObj.group(1) : ", searchObj.group(1)
   print "searchObj.group(2) : ", searchObj.group(2)
else:
   print "Nothing found!!"
```

When the above code is executed, it produces following result −

```
searchObj.group() :  Cats are smarter than dogs
searchObj.group(1) :  Cats
searchObj.group(2) :  smarter
```

Regular Expression Modifiers: Option Flags

Regular expression literals may include an optional modifier to control various aspects of matching. The modifiers are specified as an optional flag. You can provide multiple modifiers using exclusive OR (|), as shown previously and may be represented by one of these –

Sr.No.	Modifier & Description
1	**re.I** Performs case-insensitive matching.
2	**re.L** Interprets words according to the current locale. This interpretation affects the alphabetic group (\w and \W), as well as word boundary behavior(\b and \B).
3	**re.M** Makes $ match the end of a line (not just the end of the string) and makes ^ match the start of any line (not just the start of the string).
4	**re.S** Makes a period (dot) match any character, including a newline.
5	**re.U** Interprets letters according to the Unicode character set. This flag affects the behavior of \w, \W, \b, \B.
6	**re.X** Permits "cuter" regular expression syntax. It ignores whitespace (except inside a set [] or when escaped by a backslash) and treats unescaped # as a comment marker.

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 −

Sr.No.	Pattern & Description
1	**^** Matches beginning of line.
2	**$** Matches end of line.
3	**.** Matches any single character except newline. Using m option allows it to match newline as well.
4	**[...]** Matches any single character in brackets.
5	**[^...]** Matches any single character not in brackets
6	**re*** Matches 0 or more occurrences of preceding expression.
7	**re+** Matches 1 or more occurrence of preceding expression.
8	**re?** Matches 0 or 1 occurrence of preceding expression.
9	**re{ n}** Matches exactly n number of occurrences of preceding expression.
10	**re{ n,}** Matches n or more occurrences of preceding expression.

11	**re{ n, m}** Matches at least n and at most m occurrences of preceding expression.
12	**a| b** Matches either a or b.
13	**(re)** Groups regular expressions and remembers matched text.
14	**(?imx)** Temporarily toggles on i, m, or x options within a regular expression. If in parentheses, only that area is affected.
15	**(?-imx)** Temporarily toggles off i, m, or x options within a regular expression. If in parentheses, only that area is affected.
16	**(?: re)** Groups regular expressions without remembering matched text.
17	**(?imx: re)** Temporarily toggles on i, m, or x options within parentheses.
18	**(?-imx: re)** Temporarily toggles off i, m, or x options within parentheses.
19	**(?#...)** Comment.
20	**(?= re)** Specifies position using a pattern. Doesn't have a range.

21	**(?! re)** Specifies position using pattern negation. Doesn't have a range.
22	**(?> re)** Matches independent pattern without backtracking.
23	**\w** Matches word characters.
24	**\W** Matches nonword characters.
25	**\s** Matches whitespace. Equivalent to [\t\n\r\f].
26	**\S** Matches nonwhitespace.
27	**\d** Matches digits. Equivalent to [0-9].

28	**\D** Matches nondigits.
29	**\A** Matches beginning of string.
30	**\Z** Matches end of string. If a newline exists, it matches just before newline.

31	**\z** Matches end of string.
32	**\G** Matches point where last match finished.
33	**\b** Matches word boundaries when outside brackets. Matches backspace (0x08) when inside brackets.
34	**\B** Matches nonword boundaries.
35	**\n, \t, etc.** Matches newlines, carriage returns, tabs, etc.
36	**\1...\9** Matches nth grouped subexpression.
37	**\10** Matches nth grouped subexpression if it matched already. Otherwise refers to the octal representation of a character code.

Special Character Classes:

Sr.No.	Example & Description
1	**.** Match any character except newline
2	**\d** Match a digit: [0-9]
3	**\D** Match a nondigit: [^0-9]
4	**\s** Match a whitespace character: [\t\r\n\f]
5	**\S** Match nonwhitespace: [^ \t\r\n\f]
6	**\w** Match a single word character: [A-Za-z0-9_]
7	**\W** Match a nonword character: [^A-Za-z0-9_]

IP address and their classes:

Class	Address Range	Supports
Class A	1.0.0.1 to 126.255.255.254	Supports 16 million hosts on each of 127 networks
Class B	128.1.0.1 to 191.255.255.254	Supports 65,000 hosts on each of 16,000 networks
Class C	192.0.1.1 to 223.255.254.254	Supports 254 hosts on each of 2 million networks.
Class D	224.0.0.0 to 239.255.255.255	Reserved for multicast groups.
Class E	240.0.0.0 to 254.255.255.254	Reserved for future use, or research and development purposes.

Program:

```
import re

while True:

    flag=0

    print("--------------------------------------------------------------------------------")

    str=input("Enter IP address : ")

    print("--------------------------------------------------------------------------------")
```

```
    result=re.findall(r'^([0-9]|[1-9][0-9]|1[0-1][0-9]|12[0-6])[.]([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])[.]([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])[.]([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-4])$',str)
    if(len(result)!=0):
        print("Valid IP address")
        print("--------------------------------------------------------------------------------")
        print("This IP address belongs to class A")
        print("--------------------------------------------------------------------------------")
        flag=1
    result=re.findall(r'^(12[8-9]|1[3-8][0-9]|19[0-1])[.]([1-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])[.]([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])[.]([1-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-4])$',str)
    if(len(result)!=0):
        print("Valid IP address")
        print("--------------------------------------------------------------------------------")
        print("This IP address belongs to class B")
        print("--------------------------------------------------------------------------------")
```

```
        flag=1
    result=re.findall(r'^(19[2-9]|2[0-1][0-9]|22[0-3])[.]([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])[.]([1-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-4])[.]([1-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-4])$',str)
    if(len(result)!=0):
        print("Valid IP address")
        print("--------------------------------------------------------------------------------")
        print("This IP address belongs to class C")
        print("--------------------------------------------------------------------------------")
        flag=1
    result=re.findall(r'^(22[4-9]|23[0-9])[.]([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])[.]([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])[.]([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])$',str)
    if(len(result)!=0):
        print("Valid IP address")
        print("--------------------------------------------------------------------------------")
        print("This IP address belongs to class D")
```

```
        print("--------------------------------------------------------------------------------")

        flag=1

    result=re.findall(r'^(24[0-9]|25[0-4])[.]([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])[.]([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])[.]([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-4])$',str)

    if(len(result)!=0):

        print("Valid IP address")

        print("--------------------------------------------------------------------------------")

        print("This IP address belongs to class E")

        print("--------------------------------------------------------------------------------")

        flag=1

    if(flag==0):

        print("Invalid IP address!!!!!!!!")

        print("--------------------------------------------------------------------------------")

    choice=input("Do you want to continue? (Y/N) : ")

    print("--------------------------------------------------------------------------------")
```

```
    if(choice=="Y"):
        continue
    else:
        print("Thank you for using our portal!!!!!!!!")
        print("--------------------------------------------------------------------------------")
        break
```

Output:

```
--------------------------------------------------------------------------------
Enter IP address : 100.12.13.15
--------------------------------------------------------------------------------
Valid IP address
PS D:\KJSCE\OSTPL Lab> python main1.py
--------------------------------------------------------------------------------
Enter IP address : 100.12.56.89
--------------------------------------------------------------------------------
Valid IP address
--------------------------------------------------------------------------------
This IP address belongs to class A
--------------------------------------------------------------------------------
Do you want to continue? (Y/N) : Y
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Enter IP address : 129.221.45.85
--------------------------------------------------------------------------------
Valid IP address
--------------------------------------------------------------------------------
This IP address belongs to class B
--------------------------------------------------------------------------------
Do you want to continue? (Y/N) : Y
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Enter IP address : 193.45.12.78
--------------------------------------------------------------------------------
Valid IP address
--------------------------------------------------------------------------------
This IP address belongs to class C
--------------------------------------------------------------------------------
Do you want to continue? (Y/N) : Y
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Enter IP address : 225.254.254.254
--------------------------------------------------------------------------------
Valid IP address
--------------------------------------------------------------------------------
This IP address belongs to class D
--------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------
This IP address belongs to class D
--------------------------------------------------------------------------------
Do you want to continue? (Y/N) : Y
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Enter IP address : 241.224.224.224
--------------------------------------------------------------------------------
Valid IP address
--------------------------------------------------------------------------------
This IP address belongs to class E
--------------------------------------------------------------------------------
Do you want to continue? (Y/N) : Y
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Enter IP address : 256.12.12.12
--------------------------------------------------------------------------------
Invalid IP address!!!!!!!!
--------------------------------------------------------------------------------
Do you want to continue? (Y/N) : N
--------------------------------------------------------------------------------
Thank you for using our portal!!!!!!!!
--------------------------------------------------------------------------------
```

Conclusion: By performing this experiment successfully understood the concept of regular expressions in python. Also gained some knowledge about the IP addresses and their classes. Implemented a python program using regular expressions

Date: 4 / 1 / 2021 **Signature of faculty in-charge**

Post Lab Multiple Choice Questions

1. Which module in Python supports regular expressions?
a) re
b) regex
c) pyregex
d) none of the mentioned

ANS) a) re

2. Which of the following creates a pattern object?
a) re.create(str)
b) re.regex(str)

c) re.compile(str)
d) re.assemble(str)

ANS) c) re.compile(str)

3.What does the function re.match do?
a) matches a pattern at the start of the string
b) matches a pattern at any position in the string
c) such a function does not exist
d) none of the mentioned

ANS) a) matches a pattern at the start of the string

4. What does the function re.search do?
a) matches a pattern at the start of the string
b) matches a pattern at any position in the string
c) such a function does not exist
d) none of the mentioned

ANS) b) matches a pattern at any position in the string

5. What will be the output of the following Python code?

```
sentence = 'we are humans'
matched = re.match(r'(.*) (.*?) (.*)', sentence)
print(matched.groups())
```

a) (‘we’, ‘are’, ‘humans’)
b) (we, are, humans)
c) (‘we’, ‘humans’)
d) ‘we are humans’

ANS) a) (‘we’, ‘are’, ‘humans’)

6. What will be the output of the following Python code?

```
sentence = 'horses are fast'
regex = re.compile('(?P<animal>\w+) (?P<verb>\w+)
(?P<adjective>\w+)')
matched = re.search(regex, sentence)
print(matched.groupdict())
```

a) {‘animal’: ‘horses’, ‘verb’: ‘are’, ‘adjective’: ‘fast’}
b) (‘horses’, ‘are’, ‘fast’)
c) ‘horses are fast’
d) ‘are’

ANS) a) {‘animal’: ‘horses’, ‘verb’: ‘are’, ‘adjective’: ‘fast’}

7.The expression a{5} will match _____ characters with the previous regular expression.
a) 5 or less
b) exactly 5
c) 5 or more
d) exactly 4

ANS) b) exactly 5

8. _____ matches the start of the string.
_____ matches the end of the string.
a) ‘^’, ‘$’
b) ‘$’, ‘^’
c) ‘$’, ‘?’
d) ‘?’, ‘^’

ANS) a) ‘^’, ‘$’

9.What will be the output of the following Python code?

```
re.split('\W+', 'Hello, hello, hello.')
```

a) [‘Hello’, ‘hello’, ‘hello.’]
b) [‘Hello, ‘hello’, ‘hello’]
c) [‘Hello’, ‘hello’, ‘hello’, ‘.’]
d) [‘Hello’, ‘hello’, ‘hello’, ”]

ANS) d) [‘Hello’, ‘hello’, ‘hello’, ”]

10. What will be the output of the following Python function?

```
re.findall("hello world", "hello", 1)
```

a) [“hello”]
b) [ ]
c) hello
d) hello world

ANS) b) [ ]

11. Choose the function whose output can be:

<_sre.SRE_Match object; span=(4, 8), match=’aaaa’>.
a) >>> re.search(‘aaaa’, “alohaaaa”, 0)
b) >>> re.match(‘aaaa’, “alohaaaa”, 0)
c) >>> re.match(‘aaa’, “alohaaa”, 0)
d) >>> re.search(‘aaa’, “alohaaa”, 0)

ANS) a) >>> re.search(‘aaaa’, “alohaaaa”, 0)

12. Which of the following functions clears the regular expression cache?
a) re.sub()
b) re.pos()
c) re.purge()
d) re.subn()

ANS) c) re.purge()

13. What will be the output of the following Python code?

```
import re
re.ASCII
```

a) 8
b) 32
c) 64
d) 256

ANS) d) 256