Batch: A3 Roll No.: 1911034

Experiment / assignment / tutorial No.____7___

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

Signature of the Staff In-charge with date

TITLE: To check the validity of a provided IP address using regular expressions and to classify it into one among the five classes of IP's

AIM: To Understand the use of regular expressions for checking patterns in strings

Expected OUTCOME of Experiment:

CO - CO3 – To understand Regular Expressions and their usage.

Books/ Journals/ Websites referred:

1. TutorialsPoint.com
2. GeeksForGeeks.com
3. W3 Schools.com
4. Ma'am's ppt

Pre Lab/ Prior Concepts:

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.

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.

We would cover two important functions, which would be used to handle regular expressions. But a small thing first: 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'**.

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 use*group(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

Live Demo

```
#!/usr/bin/python
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

Live Demo

```
#!/usr/bin/python
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
```

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.

Program (with comments) (Write comments to your program to explain the implementation)

```
import re
ip1 = input("enter the IP address whose class is to be determined ")
flag=0
arr = ip1.split('.')
validexp1= "^[0-
9]$" #for comparing with a subpart that contains only a single digit star
ts and ends with the same digit (allowed values 0-9)
validexp2= "^[1-9][0-
9]$" #for comparing with a subpart that contains two digits starts with a
 digit from 1-9 , ends with a digit from 1-9 (allowed values 10-99)
validexp3a="^1[0-9][0-
9]$"  #for comparing with a subpart that contains three digits starts wit
h a digit from 1-2, contains a middle digit and ends with a digit from 0-
5 (allowed values 100-255)
validexp3b="^2[0-4][0-9]$"
validexp3c="^25[0-5]$"
rega="^[0-9]$|^[0-9][0-9]$|^1[0-2][0-7]$"

regb= "^1[3-8][0-9]$|^12[8-9]$|^19[0-1]$"

regc = "^19[3-9]$|^2[0-1][0-9]$|^22[0-3]$"


regd="^22[4-9]$|^23[0-9]$"
```

```
rege="^24[0-9]$|^25[0-5]$"

count=0
while count!=4: #the checking of IP address continues till any of the con
ditions is false.
    if (len(arr)!=4):
        print(len(arr))
        print("the IP address is invalid because it does not contain 4 pa
rts sepearted by a '.'")
        flag=1
        break

    else:
        for x in arr:
            if re.search("[^0-
9]",x): #checking whether each string contains digits only.
                flag=1
                print("the given input is invalid because it does not con
tain all numeric characters")
                break
            else:
                 #now we compare whether each subpart is a number between
 0 and 255
                 res2= re.search(validexp2,x)
                 res3a=re.search(validexp3a,x)
                 res3b=re.search(validexp3b,x)
                 res3c=re.search(validexp3c,x)
                 res1= re.search(validexp1,x)
                 if  res3a or res3b or res3c or res2 or res1:
                     flag=0
                     count = count+1
                 else:

                     print("not valid since each subpart does not lie bet
ween 0 and 255")
                     flag=1
                     break
        if flag==1:
            break



if flag!=1:
```

```
        if(re.search(rega,arr[0])):
            print("The IP address is valid and it's a class A IP")

        elif(re.search(regb,arr[0]) ):
            print("The IP address is valid and it's a class B IP")

        elif(re.search(regc,arr[0]) ):
            print("The IP address is valid and it's a class C IP")

        elif(re.search(regd,arr[0]) ):
            print("The IP address is valid and it's a class D IP")

        elif(re.search(rege,arr[0])  ):
            print("The IP address is valid and it's a class E IP")
```

Outputs Screenshot :

1. INVALID IP Case 1:

```
enter the IP address whose class is to be determined 23.44.55
3
the IP address is invalid because it does not contain 4 parts sepearted by a '.'
PS C:\Users\arvin\Desktop\OSTPL LAB>
```

2. INVALID IP Case 2:

```
enter the IP address whose class is to be determined 123.23.4a.57
the given input is invalid because it does not contain all numeric characters
PS C:\Users\arvin\Desktop\OSTPL LAB>
```

3. INVALID IP case 3:

```
enter the IP address whose class is to be determined 123.444.67.78
not valid since each subpart does not lie between 0 and 255
PS C:\Users\arvin\Desktop\OSTPL LAB>
```

4. Class A IP

```
regex.py"
enter the IP address whose class is to be determined 11.123.14.5
The IP address is valid and it's a class A IP
PS C:\Users\arvin\Desktop\OSTPL LAB>
```

5. Class B IP

```
enter the IP address whose class is to be determined 128.11.23.54
The IP address is valid and it's a class B IP
PS C:\Users\arvin\Desktop\OSTPL LAB>
```

6. Class C IP

```
regex.py"
enter the IP address whose class is to be determined 199.201.33.115
The IP address is valid and it's a class C IP
PS C:\Users\arvin\Desktop\OSTPL LAB>
```

7. Class D IP

```
regex.py"
enter the IP address whose class is to be determined 224.55.114.111
The IP address is valid and it's a class D IP
PS C:\Users\arvin\Desktop\OSTPL LAB>
```

8. Class E IP

```
regex.py
enter the IP address whose class is to be determined 251.55.222.115
The IP address is valid and it's a class E IP
PS C:\Users\arvin\Desktop\OSTPL LAB>
```

Conclusion: The concepts of Regular expressions was well understood and implemented to validate and IP address and classify it into the different classes.

Date: _____ **Signature of faculty in-charge**