Programming for Data Science
CSE3041

Prof. Ramesh Ragala

July 30, 2020

Problem - 1

Write a Python code to check if the given mobile number is valid or not. The conditions to be satisfied for a mobile number are:

a) Number of characters must be 10

b) All characters must be digits and must not begin with a '0'

Test Case -1

- abc8967891

Test Case -1

- abc8967891
- Invalid
- Alphabets are not allowed

Test Case -1

- abc8967891
- Invalid
- Alphabets are not allowed

Test Case -2

- 440446845

Test Case -1

- abc8967891
- Invalid
- Alphabets are not allowed

Test Case -2

- 440446845
- Invalid
- Only 9 Digits

Test Case -3

- 0440446845

Test Case -3

- 0440446845
- Invalid
- Should not begin with a zero

Test Case -3

- 0440446845
- Invalid
- Should not begin with a zero

Test Case -4

- 8440446845

Test Case -3

- 0440446845
- Invalid
- Should not begin with a zero

Test Case -4

- 8440446845
- Valid
- All conditions are satisfied

Python code to check validity of mobile number (Long Code)

```
import sys
number = input("Enter Mobile Number")
if len(number)!=10:
    print ('invalid')
    sys.exit(0)
if number[0]=='0':
        print ('invalid')
        sys.exit(0)
for chr in number:
    if chr.isalpha():
        print ('invalid')
        break
else:
    print('Valid')
```

Problem -2

If we are running an e-mail archiving company, and you, as one of my customers, requested all of the e-mail that you sent and received last February, for example, it would be nice if I could set a computer program to collate and forward that information to you, rather than having a human being read through your e-mail and process your request manually.

Problem - 3

A request might be to look for a subject line like WINGOLD, indicating a virus-infected message, and remove those e-mail messages from your personal archive.

- Manipulating text or data is a complex thing.
- So the above examples demands the question of how we can program machines with the ability to look for patterns in text
- **Regular expressions** provide such an infrastructure for advanced text pattern matching, extraction, and/or search-and-replace functionality.
- Python supports regexes through the standard library **re** module.
- regexes are strings containing **text and special characters** that describe a **pattern** with which to recognize multiple strings.

- Regexs without special characters

Regex Pattern	String(s) Matched
foo	foo
Python	Python
abc123	abc123

- These are simple expressions that match a single string.
- Power of regular expressions comes in when special characters are used to define character sets, subgroup matching, and pattern repetition.

Special Symbols and Characters

Notation Symbols	Description	Example Regex
literals	Match literal string value literal	foo
re1 | re2	Match regular expression re1 or re2	foo | bar
.	Match any character except \n	b.b
^	Match start of the string	^Dear
$	Match end of string	/bin/*sh$
*	Match 0 or more occurrences of preceding regex	$[A-Za-z0-9]*$
+	Match 1 or more occurrences of preceding regex	$[a-z]+\.com$
?	Match 0 or 1 occurrence(s) of preceding regex	goo?

Special Symbols and Characters - Contd...

Notation Symbols	Description	Example Regex
{N}	Match N occurrences of preceding regex	$[0-9]\{3\}$
{M,N}	Match M to N occurrences of preceding regex	$[0-9]\{5,9\}$
[...]	Match from M to N occurrences of preceding regex	$[aeiou]$
[..x−y..]	Match any single character in the range from x to y	$[0-9], [A-Za-z]$
[^]	Do not match any character from character class, including any ranges, if present	$[\hat{}aeiou]*[\hat{}A-Za-z0-9]$

- **Matching any single Character(.):**
  - Dot or Period (.) symbol (letter, number, whitespace (not including "\n"), printable, non-printable, or a symbol) matches any single character except for '\n'.
  - To specify a dot character explicitly, you must escape its functionality with a backslash, as in "\."

Regex Pattern	Strings Matched
f.o	Any Character between "f " and "o"; for example:- fao, f9o, f#0 etc..
..	Any pair of characters
.end	Any character before the string end

- The re.search() method takes a regular expression pattern and a string and searches for that pattern within the string
- If the search is successful, search() returns a match object or None otherwise

Example-1

```
import re
if re.match("f.o","fooo"):
    print("Matched")
else:
    print("Not matched")
```

Output:

Prints matched

Since it searches only for the pattern f.o in the string

Example-2

```
import re
if re.match("f.o$","fooo"):
    print("Matched")
else:
    print("Not matched")
```

Check that the entire string starts with 'f', ends with 'o' and contain one letter in between

Example-3

```
import re
if re.match("..","fooo"):
    print("Matched")
else:
    print("Not matched")
```

Matched

Two dots matches any pair of characters.

Example-4

```
import re
if re.match("..$","fooo"):
    print("Matched")
else:
    print("Not matched")
```

Not matched

Including a '$' at the end will match only strings of length 2

Example-5

```
import re
if re.match(".end","bend"):
    print("Matched")
else:
    print("Not matched")
```

Matched

The expression used in the example, matches any character for '.'

Example-6

```
import re
if re.match(".end","bends"):
    print("Matched")
else:
    print("Not matched")
```

Prints Matched

The expression used in the example, matches any character for '.'

Example - 7

```
import re
if re.match(".end$","bends"):
    print("Matched")
else:
    print("Not matched")
```

Prints Not matched - $ check for end of string

Matching from the Beginning or End of Strings or Word Boundaries (^, $)

- ^ - Match beginning of string
- $ - Match End of string

Regex Pattern	Strings Matched
^From	Any string that start with From
/bin/tcsh$	Any String that ends with /bin/tcsh
^Subject: hi$	Any String consisting solely of the string Subject: hi

- if you wanted to match any string that ended with a dollar sign, one possible regex solution would be the pattern .*\$$

But not sufficient

- Check whether the given register number of a VIT student is valid or not.
- Example register number - 16bec1032
- Register number is valid if it has two digits
- Followed by three letters
- Followed by four digits

Denoting Ranges (-) and Negation (^)

- brackets also support ranges of characters
- A hyphen between a pair of symbols enclosed in brackets is used to indicate a range of characters;
- For example A-Z, a-z, or 0-9 for uppercase letters, lowercase letters, and numeric digits, respectively

Regex Pattern	Strings Matched
z,[0 – 9]	"z" followed by any character then followed by any single digit
[r-u] [env-y] [us]	"r","s","t" or "u" followed by "e","n","v","w","x" or "y" followed by "u" or "s"
^[*aeiou*]	A non-vowel character (Exercise: Why do we say "non-vowels" rather than "consonants"
[^\t\n]	Not a TAB or \n
["-a]	In an ASCII system, that all characters fall between ' " ' and 'a', i.e., between ordinals 34 and 97

Multiple Occurrence/Repetition Using Closure Operators (*, +, ?, {})

- respecial symbols *, +, and ?, all of which can be used to match single, multiple, or no occurrences of string patterns
- Asterisk or star operator (*) - match zero or more occurrences of the regex immediately to its left
- Plus operator (+) - Match one or more occurrences of a regex
- Question mark operator (?) - match exactly 0 or 1 occurrences of a regex.
- There are also brace operators ({}) with either a single value or a comma-separated pair of values. These indicate a match of exactly N occurrences (for {N}) or a range of occurrences; for example, {M, N} will match from M to N occurrences

Code to check the validity of register number

```
import re
register= input()
if re.match("^[1-9][0-9][a-zA-Z][a-zA-Z][a-zA-Z][0-9][0-9][0-9][0-9] $",register):
    print("Matched")
else:
    print("Not matched")
```

^ - denote begin (Meaning is different when we put this symbol inside the square bracket)

$ - denote end

Regex Pattern	Strings Matched
[*dn*]ot?	"d" or "n", followed by an "o" and, at most one "t" after that; do, not, dot, not.
0?[1 – 9]	Any numeric digit, possibly prepended with digit "0". For example the numeric representations of months January to September, Whether single or double digit.
[0 – 9]{15, 16}	Fifteen or Sixteen Digits (Example: Credit Card Number)

$\{n\}$ indicate that the pattern before the braces should occur n-times.

Refined Code to check the validity of register number

```
inport re
register = input()
if re.match("^[1 9][0 9][a zA Z]{3}[0 9]{4}",register)
    print("Matched")
else:
    print("Not Matched")
```

Checking the format of the mobile number

```
import re
number = input()
if re.match("[^0][0 9]{9}",number):
    print("valid Mobile Number")
else:
    print("Invalid Mobile Number")
```

Bug: It will also accept R8097488270

Checking the format of the mobile number

```
import re
number = input()
if re.match("[1 9][0 9]{9}",number):
    print("valid Mobile Number")
else:
    print("Invalid Mobile Number")
```

Checking the format of the PAN number using Regular Expression

```
import re
pan=input()
if len(pan) < 10 and len(pan) > 10 :
    print ("PAN Number should be 10 characters")
    exit
elif re.search("[^a zA Z0 9]",pan):
    print ("No symbols allowed, only alphanumerics")
    exit
elif re.search("[0 9]",pan[0:5]):
    print ("Invalid  1")
    exit
elif re.search("[A Za z]",pan[5:9]):
    print ("Invalid  2")
    exit
elif re.search("[0 9]",pan[ 1]):
    print ("Invalid  3")
    exit
else:
    print ("Your card "+ pan + " is valid")
```

- Python read all input as string
- In some cases it is necessary to check if the value entered is an integer $\rightarrow$ we can check it using regular expressions
- **Rules for an integer**
  - optionally begin with a negative sign include ^ symbol
  - first digit must be a number other than zero
  - may be followed zero to any number of digits
  - string must end with it so add $ symbol

Checking the format of the Integer Number:

```
import re
register= input()
#optionally begin with a negative sign include ^ symol
#first digit must be a number other than zero
# may be followed zero to any number of digits
# string must end with it so add $ symbol
if re.match("^\?[1-9][0-9]*$",register):
#'\' is added in front of '?' to overcome
#its default meaning in REs
    print("Matched")
else:
    print("Not matched")
```

- **Rules for an integer or floating point number**
  - optionally begin with a negative sign include ^ symbol
  - first digit must be a number other than zero
  - may be followed zero to any number of digits
  - string must end with it so add $ symbol
  - Optionally followed by a '.'
  - Followed by zero or more digits
  - String ends here

Checking the format of the Integer or floating point values:

```
import re
register= input()
if re.match("^\ ?[1 9][0 9]*\.?[0 9]*$",register):
# '.' can occur zero or one time followed by a digit
#occurred zero to infinite number of times
    print("Matched")
else:
    print("Not matched")
```

- Example program to searches for the pattern 'word:' followed by a 3 letter word

Example-1

```
import re
str1 = input('Enter a string')
match = re.search(r'word:\w\w\w', str1)
# If statement after search() tests if it succeeded
if match:
    print('found', match.group())## 'found word:cat'
else:
    print('did not find')
```

Example-2

```
import re
str1 = 'piiig'
d1 = re.search(r'iii',str1)
d1.group();
if(d1):
    print('Found -> ',d1.group())
else:
    print('Not Found')

d2 = re.search(r'igs',str1)
if(d2):
    print('Found -> ',d2.group())
else:
    print('Not Found')
```

Example-3

```
import re
str1 = 'piiig'
d3 = re.search(r'.g',str1)
s4=d3.group()
if(s4):
    print('Found -> ',s4)
else:
    print('Not Found')

str2 = 'vit123uni'
d4 = re.search(r'\d\d\d',str2)
s5=d4.group()
if(s5):
    print('Found -> ',s5)
else:
    print('Not Found')
```

Example - 4

```
import re # \w matches a word character:
#a letter or digit or underbar
#[a zA Z0 9 _]
str1 = '@@##abc123@@&'
d3 = re.search(r'\w\w\w',str1)
s4=d3.group()
if(s4):
    print('Found -> ',s4)
else:
    print('Not Found')
# check str1=@@##9abcd123@##'
```

Example - 5

```
import re
# one or more occurrences of the pattern
# to its left
str1 = 'piiiiiiiiig'
d3 = re.search(r'pi+',str1)
s4=d3.group()
if(s4):
    print('Found > ',s4)
else:
    print('Not Found')
```

Example - 6

```
import re
# finds the leftmost word
str1 = 'piiigiiiiiii'
d3 = re.search(r'i+',str1)
s4=d3.group()
if(s4):
    print('Found -> ',s4)
else:
    print('Not Found')
```

Example - 7

```
import re
# looking for 3 digits, possibly
#separated by whitespace
# \s   > whitespace characters
# \s*   > zero or more whitespace chars
str1 = 'VIT1 2  3UNIVERSITY'
str2 = 'VIT12 3UNIVERSITY'
str3 = 'VIT123UNIVESITY'
d3 = re.search(r'\d\s*\d\s*\d',str1)
print(d3.group())
d4 = re.search(r'\d\s*\d\s*\d',str2)
print(d4.group())
d5 = re.search(r'\d\s*\d\s*\d',str3)
print(d5.group())
```

Example - 8

```
import re
# ^= matches the start of string, so this fails:
str1 = 'University'
str2 = 'VITUNIVERSITY'
d3 = re.search(r'^U\w+',str1)
print(d3.group())
d4 = re.search(r'^U\w+',str2)
#d4.group()
if (d4):
    print('found')
else:
    print('Not Found')
d5=re.search(r'S\w+',str2)
print(d5.group())
```

Example - 9

```
import re
# want to find the email id in given strings
str1 = input('Enter a string including email')
d3 = re.search(r'\w+@\w+',str1)
if(d3):
    print(' email id found > ',d3.group())
else:
    print('no maild id found')
# try ramesh.ragala@vit.ac.in
```

Example - 10

```
import re
# []   > used to indicate the set of chars,
#[abc]   > matches a or b or c \w and \s also works ex
str1 = input('Enter a string including email')
d3 = re.search(r'[\w.]+@[\w.]+',str1)
if(d3):
    print(' email id found > ',d3.group())
else:
    print('no maild id found')
#try ramesh_ragala@yahoo.co.in
```

Example - 11

```
import re
# ()   > used for group feature for regular expression
# pick ups the parts of the string
# Extracts the username and domain name separately
str1 = input('Enter a string including email')
d3 = re.search(r'([\w.]+)+@([\w.]+)',str1)
if(d3):
    print(' email id found > ',d3.group())
    print('username of the mail id > ', d3.group(1))
    print('domain name of the mailid > ',d3.group(2))
else:
    print('no maild id found')
```

- **findall():**
  - it is power function in re module
  - it is used to find all matches and stores as list of strings

Example - 11

```
import re
str1 = input('Enter a string')
d4 = re.findall(r'v.t',str1)
for ch in d4:
    print(ch)
```

Example - 12

```
import re
str1 = input('Enter a string')
d4 = re.findall(r'[\w.]+@[\w\.]+',str1)
for mailid in d4:
    print(mailid)
```


thank you