re.search("csci","csci538")
re.search("\d\d\d","csci538")

Regular Expression (Regex)

Dr. Yuehua Wang

yuehua.wang@tamuc.edu

TEXAS A&M UNIVERSITY COMMERCE

import re

Whaaaaaaat??????

https://www.youtube.com/watch?time_continue=23&v=zN8rwVXwRUE&feature=emb_logo
https://www.regular-expressions.info/tutorialcnt.html

What we will study

- What is a Regular Expression?
- Why we use Regular Expressions
  - Examples:
    - Find out interested info
    - Verify email addresses
    - Verify phone numbers and country code
    - Find the interested info and replace it
- Check its compatibility
- Examine basic regular expression operations
- Applications

What is a Regular Expression?

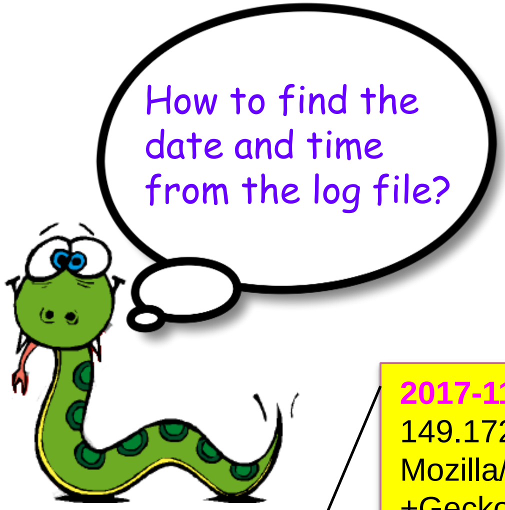
- A Regular Expression is a special text string for describing a search pattern.
  - It is a technique developed in theoretical computer science and formal language theory.
  - Initially, it was most widely used with Perl. Now, most programming languages provide regex capabilities either built-in or via libraries.

```
import re
re.search(“csci”,”csci538”)
re.search(“\d\d\d”,”csci538”)
```

Whaaaaaaat??????

Why we use Regular Expressions

- Example 1: find out interested info

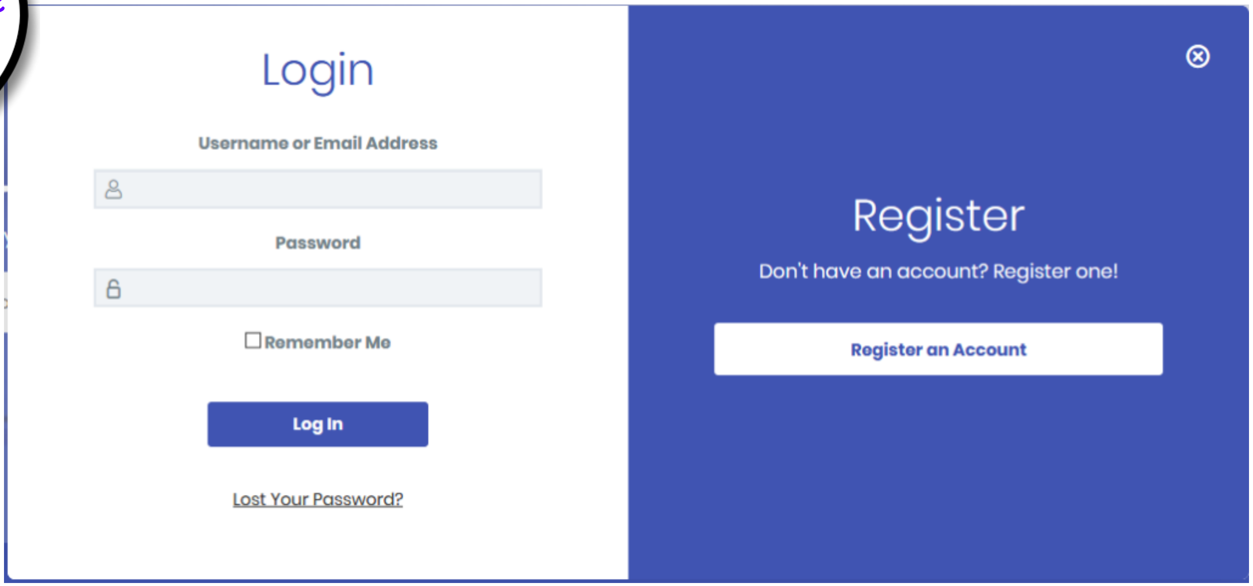


```
2017-11-18 08:48:20 GET /de/ adpar=12345&gclid=1234567890 443 -
149.172.138.41 HTTP/2.0
Mozilla/5.0+(Windows+NT+10.0;+Win64;+x64)+AppleWebKit/537.36+(KHTML,+like
+Gecko)+Chrome/62.0.3202.89+Safari/537.36+OPR/49.0.2725.39 -
https://www.google.de/ www.site-logfile-explorer.com 200 0 0 12973 544 62
2017-11-18 11:45:11 GET /global/lwb.min.js - 443 - 87.185.206.252 HTTP/2.0
Mozilla/5.0+(Windows+NT+10.0;+Win64;+x64;+rv:57.0)+Gecko/20100101+Firefox/
57.0 _ga=GA1.2.573603466.1510956966;+_gid=GA1.2.622072548.1510956966
https://translate.google.com/ www.site-logfile-explorer.com 200 0 0 2429 473 15
```

```
2017-11-18 08:48:20
2017-11-18 11:45:11
```

date and time

- Example 2: verify email addresses


Login

Username or Email Address

Password

□ Remember Me

Register

Don't have an account? Register one!

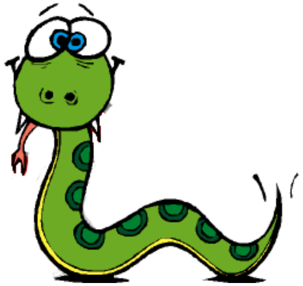
Sc # .com
sk@gmail1.com
dk@gmail.com
s@k2@ga,cl.com
Cf # 4 @ c m

- Example 3: verify the phone number and find the country to which it belongs

+4
+1
+3
+7
+8
+9
+2
+6
+5
1
7
86
91
61
52
57
51
55
56
54
58
90
98
93
92
95
66
60
62
63
64
82
81
20
27
33
34
39
46
48

444-123-2344
10-102904562
109-2937-034
100-2939-9390

- Example 4: find the interested info and replace it



Student ID: 001
Name: Jessica
DoB: Mar-03-2000
Phone: 765-098-3455
Address: 805 S Arlington Blvd,
Apt 1A, Arlington VA 22200

Student ID: …….
……

Student ID: …….
……

22200
22200
22200
22200
22200

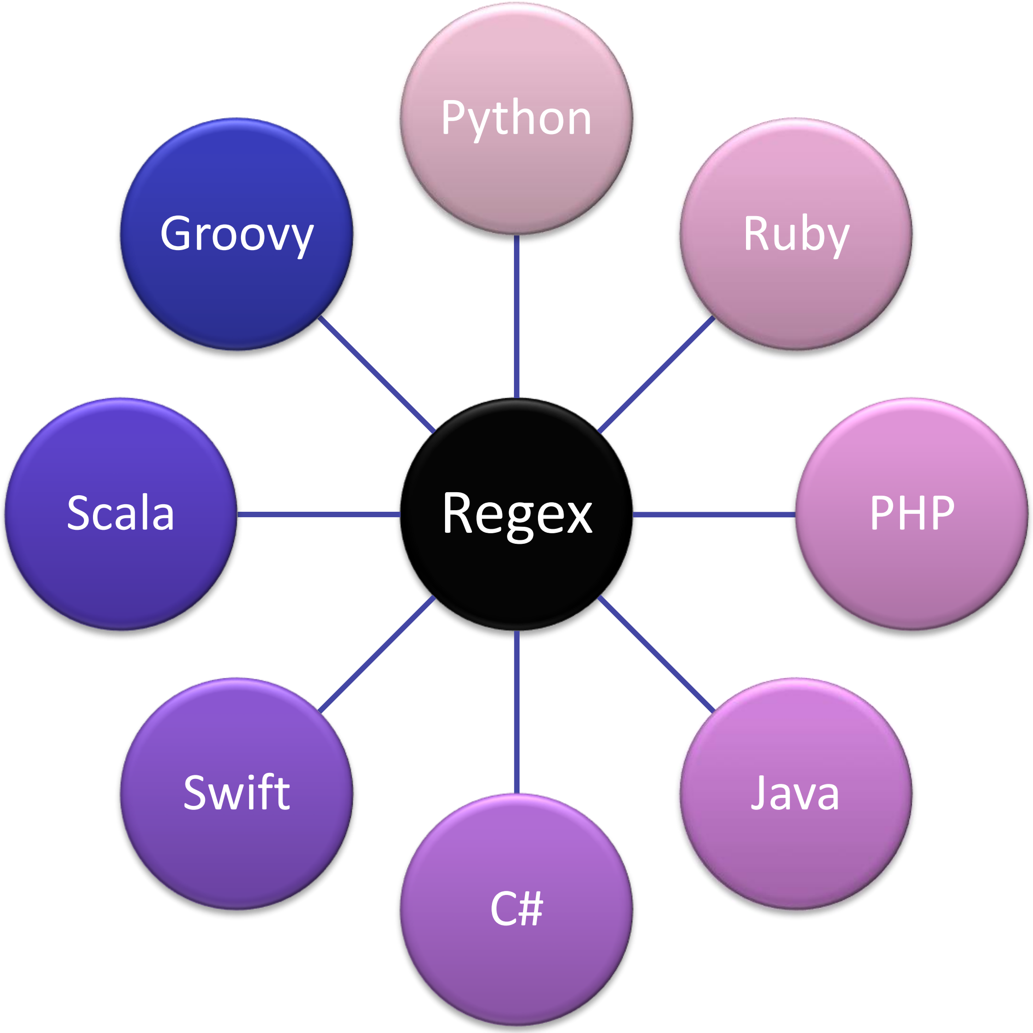
find the strings

75428
75428
75428
75428
75428

Replace the strings

- Regex is especially useful for text processing tasks
    - Specifically, you can use them to:
        - find text that matches the pattern within a larger body of text,
        - verify whether input fits into the text pattern,
        - replace text matching the pattern with other text or rearranged bits of the matched text,
        - split a block of text into a list of subtexts, among other things.

Compatible with different languages



HOW TO REGEX
STEP 1: OPEN YOUR FAVORITE EDITOR
@GARABATOKID
STEP 2: LET YOUR CAT PLAY ON YOUR KEYBOARD
/^([A-Z0-9_\.-

- Set your programming environment
  - pip install regex
- Open your editor
  - Jupyter notebook

RE Module

- Regex can be used in most programming languages. In Python the “re” module provides regex support.
- Simply, import re
- Main functions in re module:
  - re.match(A, B) : Search the regular expression and return the first occurrence
  - re.search(A, B) : Matches the first instance of an expression A in a string B, and returns it as a re match object
  - re.findall(A,B) : Matches all instances of an expression A in a string B and returns them in a list
  - re.sub(A, B, C) : Replace A with B in the string C

re.search() vs re.match()

- Use of re.search() and re.match()
  - re.search() and re.match() both are functions of re module in python. These function are very efficient and fast for searching in strings. The function searches for some substring in a string and return a match object if found, else it returns none.
- re.search() vs re.match()
  - There is a difference between the use of both functions. Both return first match of a substring found in the string, but
    - re.match() searches only in the first line of the string and return match object if found, else return none. But if a match of substring is found in some other line other than the first line of string (in case of a multi-line string), it returns none.
    - While re.search() searches for the whole string even if the string contains multi-lines and tries to find a match of the substring in all the lines of string.

```
# import re module
import re

subString ='string’


string = '''We are learning regex in CSCI538
           regex is very useful for string matching.
           It is fast too.'''

# Use of re.search() Method
print(re.search(subString, string, re.IGNORECASE))

# Use of re.match() Method
print(re.match(subString, string, re.IGNORECASE))
```

- Each character in a regular expression is either:
  - a metacharacter, having a special meaning
    - the backslash \, the caret ^, the dollar sign $, the period or dot ., the vertical bar or pipe symbol |, the question mark ?, the asterisk or star *, the plus sign +, the opening parenthesis (, the closing parenthesis), the opening square bracket [, and the opening curly brace {
  - a regular character that has a literal meaning.
    - All characters except the listed special characters match a single instance of themselves
    - Examples: ‘a’, ‘pre’, or ‘less’

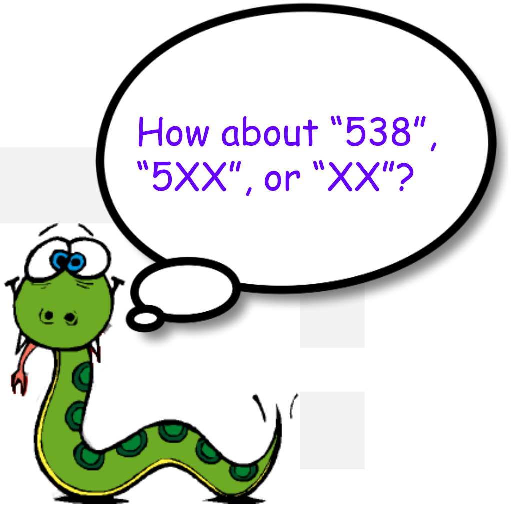
Basic regular expression operations

- Write Python code to find out whether string contains the substring ‘csci’

```
string ="csci 538"
s="csci"
s in string
```

Output:

```
True
```



```
string.find(s)
```

Output:

```
0
```

```
string.index(s)
```

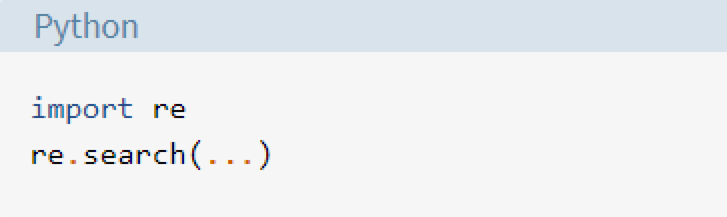
```
0
```

- import the python library 're' for regular expressions

```
#import the python library 're' for regular expressions
import re
```

Import re.search()

- Use the module name 're' as a prefix when calling the function



Python

```
import re
re.search(...)
```

- First Pattern-Matching Example

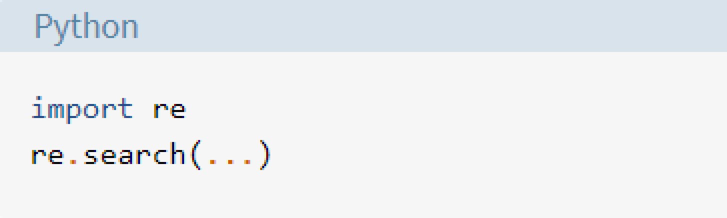
```
import re
string ="csci 538"
s="csci"
re.search(s,string)
```

Output:

```
<_sre.SRE_Match object; span=(0, 4), match='csci'>
```

Import re.search()

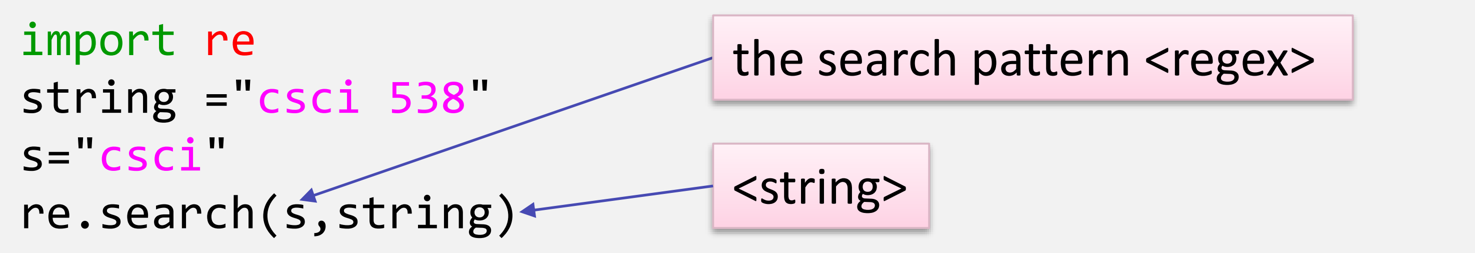
- Use the module name 're' as a prefix when calling the function



Python

```
import re
re.search(...)
```

- First Pattern-Matching Example

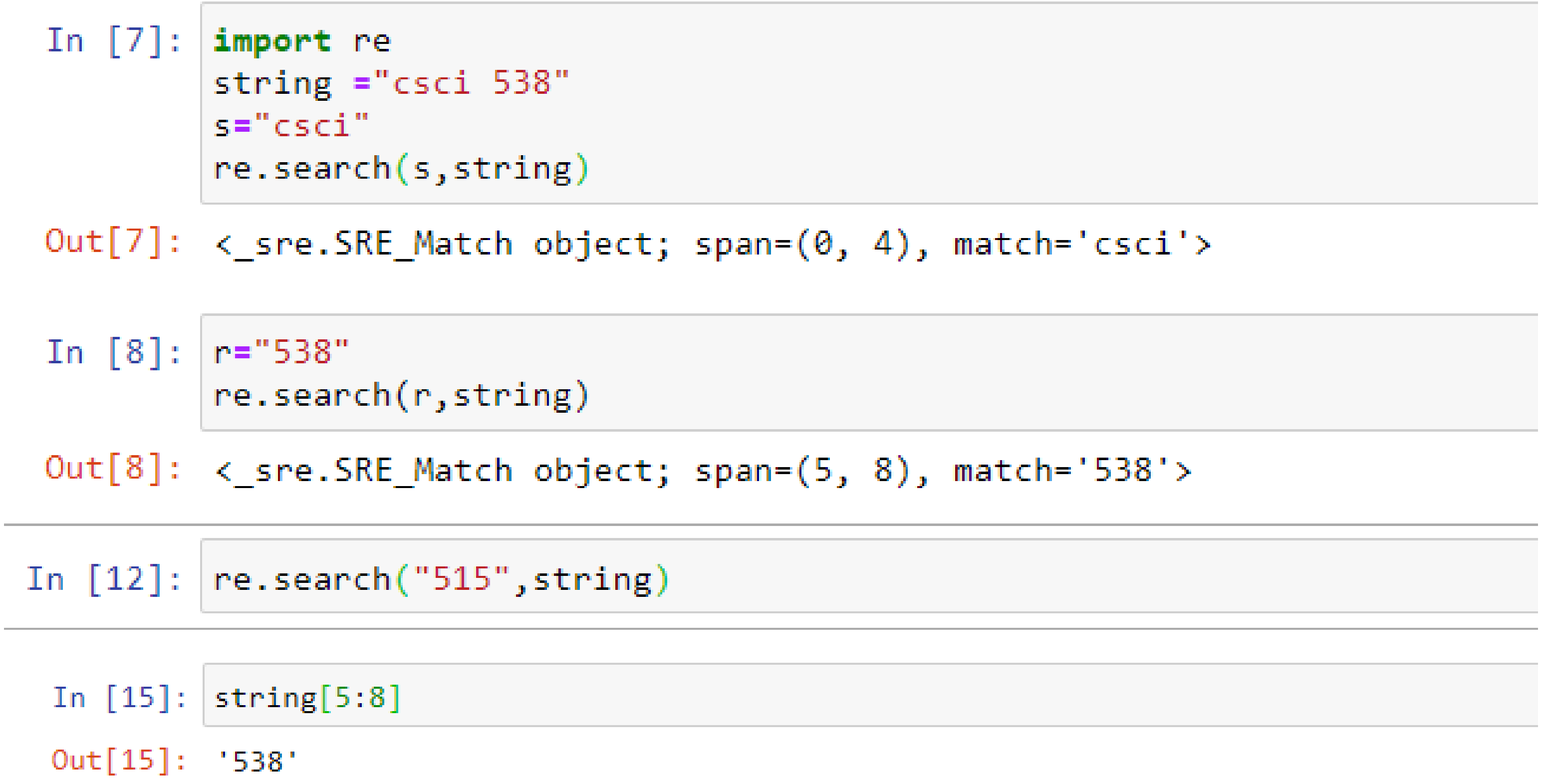


```
import re
string ="csci 538"
s="csci"
re.search(s,string)
```

Output:

```
<_sre.SRE_Match object; span=(0, 4), match='csci'>
```

First Pattern-Matching Example



```
In [7]:   import re
          string ="csci 538"
          s="csci"
          re.search(s,string)

Out[7]:   <_sre.SRE_Match object; span=(0, 4), match='csci'>

In [8]:   r="538"
          re.search(r,string)

Out[8]:   <_sre.SRE_Match object; span=(5, 8), match='538'>

In [12]:  re.search("515",string)

In [15]:  string[5:8]

Out[15]:  '538'
```

- A match object is **truthy**, so you can use it in a Boolean context like a conditional statement:

```
if re.search('538’, string):
    print('Found a match.')
else:
    print('No match.')
```

Output:

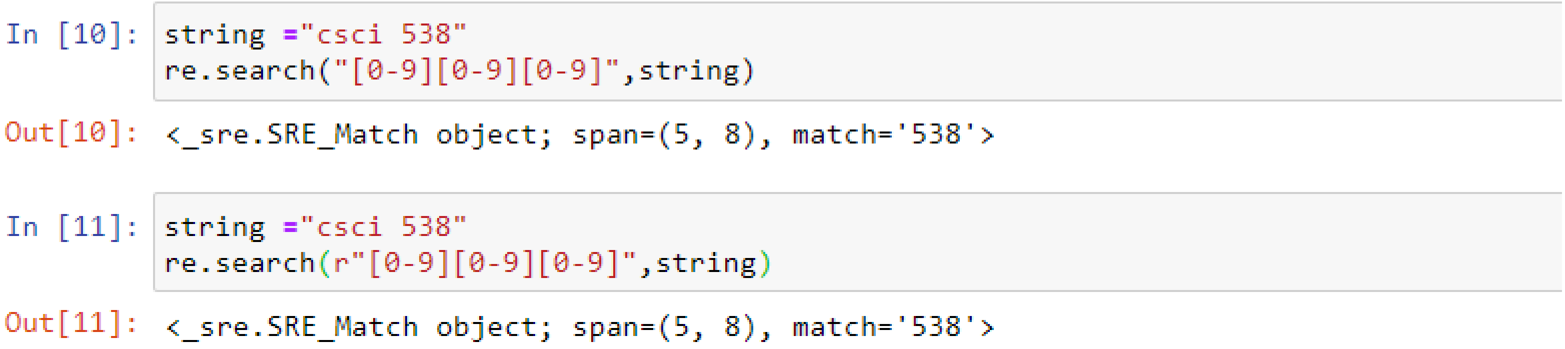
```
Found a match.
```

Python Regex Metacharacters

- The real power of regex matching in Python emerges when <regex> contains special characters (metacharacters).

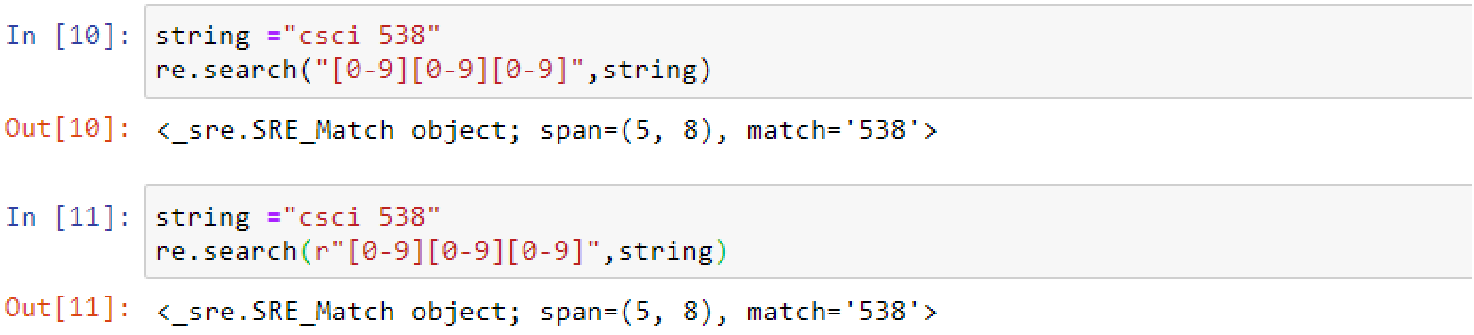
- These have a unique meaning to the regex matching engine and vastly enhance the capability of the search.

- Consider again the problem of how to determine whether a string contains any three consecutive decimal digit characters.



```
In [10]: string ="csci 538"
         re.search("[0-9][0-9][0-9]",string)

Out[10]: <_sre.SRE_Match object; span=(5, 8), match='538'>

In [11]: string ="csci 538"
         re.search(r"[0-9][0-9][0-9]",string)

Out[11]: <_sre.SRE_Match object; span=(5, 8), match='538'>
```

- Consider again the problem of how to determine whether a string contains any three consecutive decimal digit characters.

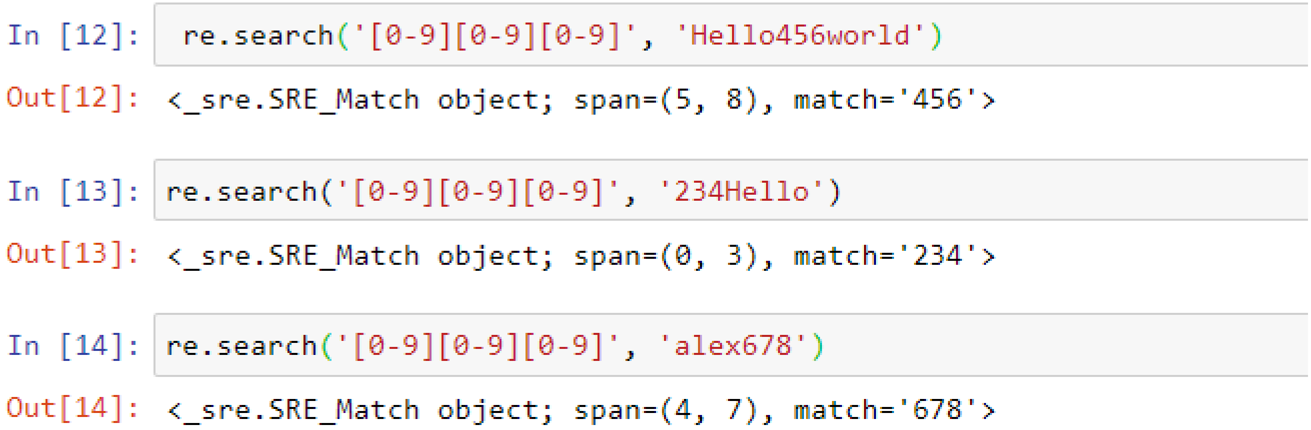


```
In [10]: string ="csci 538"
         re.search("[0-9][0-9][0-9]",string)

Out[10]: <_sre.SRE_Match object; span=(5, 8), match='538'>

In [11]: string ="csci 538"
         re.search(r"[0-9][0-9][0-9]",string)

Out[11]: <_sre.SRE_Match object; span=(5, 8), match='538'>
```

  - In a regex, a set of characters specified in square brackets ([]) makes up a character class.
  - [0-9] matches any single decimal digit character—any character between '0' and '9', inclusive.
  - The full expression [0-9][0-9][0-9] matches any sequence of three decimal digit characters.

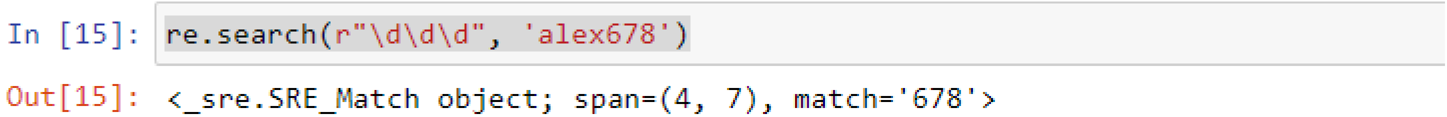
- More examples,

```
re.search(‘[0-9][0-9][0-9]’, ‘Hello456world’)
re.search('[0-9][0-9][0-9]', '234Hello’)
re.search('[0-9][0-9][0-9]', 'alex678')
```



```
In [12]: re.search('[0-9][0-9][0-9]', 'Hello456world')

Out[12]: <_sre.SRE_Match object; span=(5, 8), match='456'>

In [13]: re.search('[0-9][0-9][0-9]', '234Hello')

Out[13]: <_sre.SRE_Match object; span=(0, 3), match='234'>

In [14]: re.search('[0-9][0-9][0-9]', 'alex678')

Out[14]: <_sre.SRE_Match object; span=(4, 7), match='678'>
```

- With regexes in Python, you can identify patterns in a string that you wouldn’t be able to find with the in operator or with string methods.



```
In [15]: re.search(r"\d\d\d", 'alex678')

Out[15]: <_sre.SRE_Match object; span=(4, 7), match='678'>
```

Metacharacters Supported by the re Module

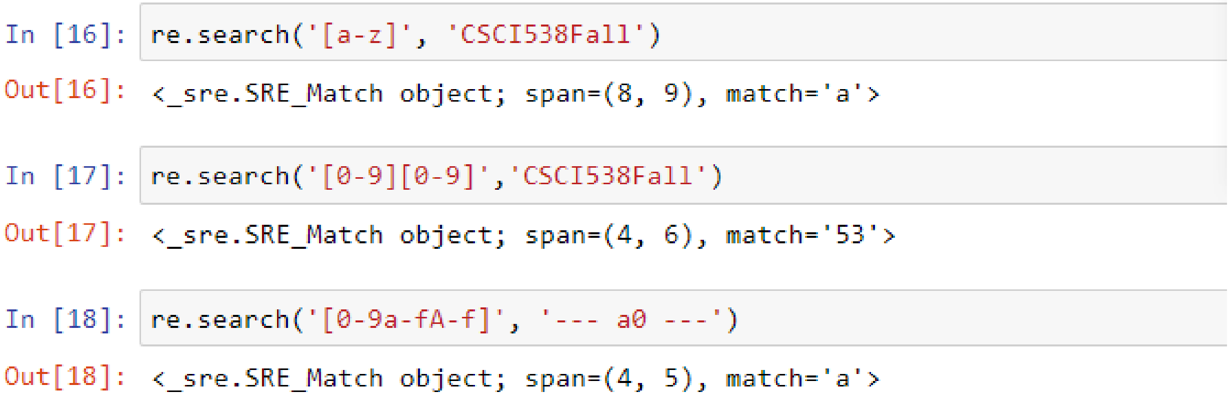
- The following table briefly summarizes all the metacharacters supported by the re module. Some characters serve more than one purpose:

Character	Meaning
.	Matches any single character except newline
^	• Anchors a match at the start of a string • Complements a character class
$	Anchors a match at the end of a string
*	Matches zero or more repetitions
+	Matches one or more repetitions
?	• Matches zero or one repetition • Specifies the non-greedy versions of *, +, and ? • Introduces a lookahead or lookbehind assertion • Creates a named group
{}	Matches an explicitly specified number of repetitions
\	• Escapes a metacharacter of its special meaning • Introduces a special character class • Introduces a grouping backreference
[]	Specifies a character class
|	Designates alternation
()	Creates a group
: # = !	Designate a specialized group
<>	Creates a named group

Character	Meaning
\d	Matches a digit in 0-9
\D	Matches a non-digit(anything but not a digit)
\w	Matches a word like a letter, digit, or underscore(_)
\W	Matches a non-word
\s	Matches a whitespace (space,\t,\r, or\n)
\S	Matches a non-whitespace

- A character class can also contain a range of characters separated by a hyphen (-), in which case it matches any single character within the range.
    - For example, [a-z] matches any single lowercase alphabetic character between 'a' and 'z', inclusive:

```
re.search('[a-z]', 'CSCI538Fall’)
re.search('[0-9][0-9]','CSCI538Fall’)
re.search('[0-9a-fA-f]', '--- a0 ---')
```

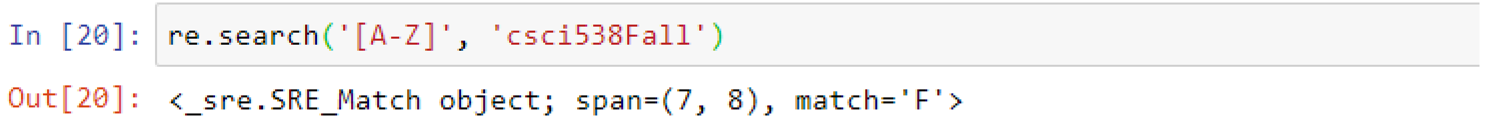


```
In [16]: re.search('[a-z]', 'CSCI538Fall')
Out[16]: <_sre.SRE_Match object; span=(8, 9), match='a'>

In [17]: re.search('[0-9][0-9]','CSCI538Fall')
Out[17]: <_sre.SRE_Match object; span=(4, 6), match='53'>

In [18]: re.search('[0-9a-fA-f]', '--- a0 ---')
Out[18]: <_sre.SRE_Match object; span=(4, 5), match='a'>
```

- Note: In the above examples, the return value is always the leftmost possible match (first occurance).
  - re.search() scans the search string from left to right, and as soon as it locates a match for <regex>, it stops scanning and returns the match.

```
re.search('csc[aeiou]', 'csci538Fall')
```

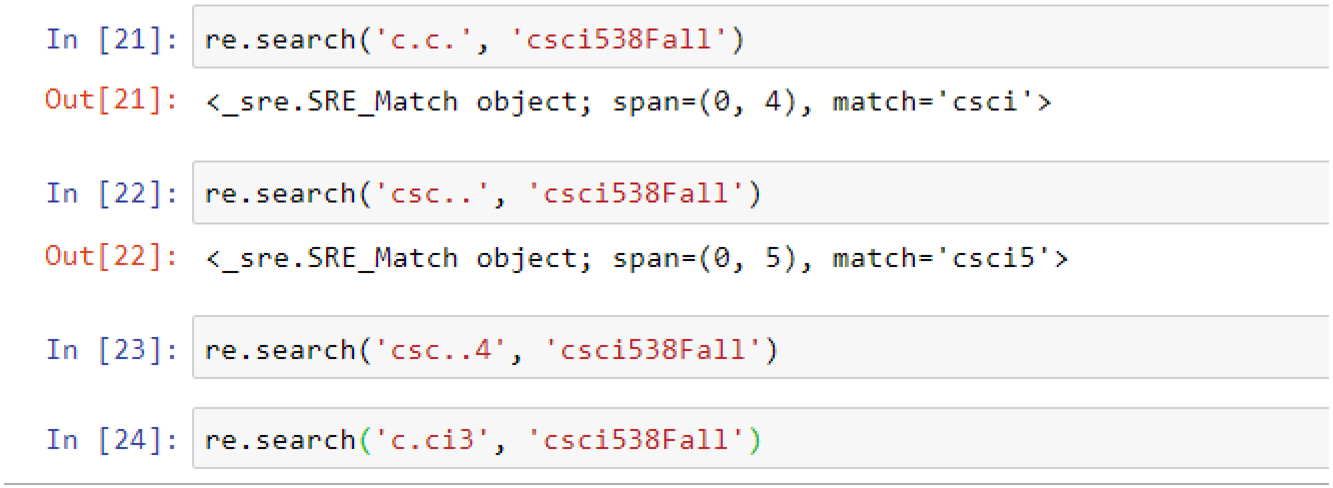
  - The metacharacter sequence [aeiou] matches any single 'a', ‘e', ‘i’,‘o’, or 'z' character. In the example, the regex csc[aeiou] matches both ‘csca’,’csce' (and would also match ‘csci’,‘csco’, and ‘cscu’).



```
In [20]: re.search('[A-Z]', 'csci538Fall')
Out[20]: <_sre.SRE_Match object; span=(7, 8), match='F'>
```

Metacharacters – dot (.)

- dot (.) specifies a wildcard
  - Matches any single character except a newline

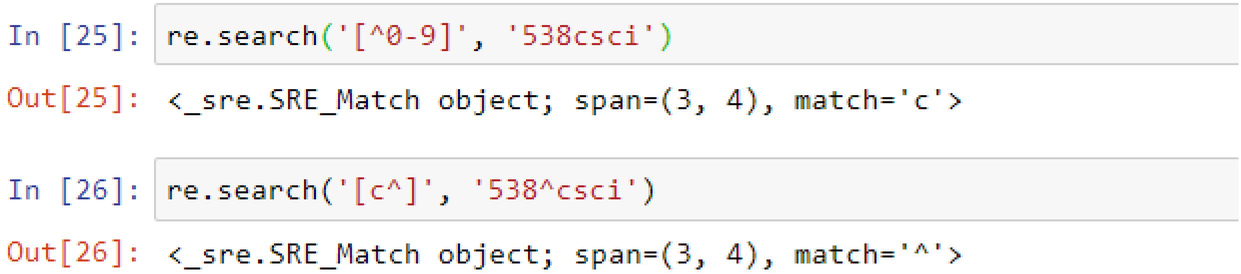


```
In [21]: re.search('c.c.', 'csci538Fall')
Out[21]: <_sre.SRE_Match object; span=(0, 4), match='csci'>

In [22]: re.search('csc..', 'csci538Fall')
Out[22]: <_sre.SRE_Match object; span=(0, 5), match='csci5'>

In [23]: re.search('csc..4', 'csci538Fall')

In [24]: re.search('c.ci3', 'csci538Fall')
```

  - As a regex, csc essentially means the characters ‘csc', then (.) any character except newline, other dot (.)
  - The first string shown above, ‘csci', fits the bill because the . metacharacter matches the ‘i'.

Metacharacters - caret (^)

- caret (^) : if it is the first character in the character class [], it specifies any character that isn't in the set.
  - For example, [^0-9] matches any character that isn't a digit



```
In [25]: re.search('[^0-9]', '538csci')
Out[25]: <_sre.SRE_Match object; span=(3, 4), match='c'>

In [26]: re.search('[c^]', '538^csci')
Out[26]: <_sre.SRE_Match object; span=(3, 4), match='^'>
```

    - [25] - the match object indicates that the first character in the string that isn't a digit is 'c'.
    - [26] – the match object with c or ^
      - If a ^ character appears in a character class but isn't the first character, then it has no special meaning and matches a literal '^' character:

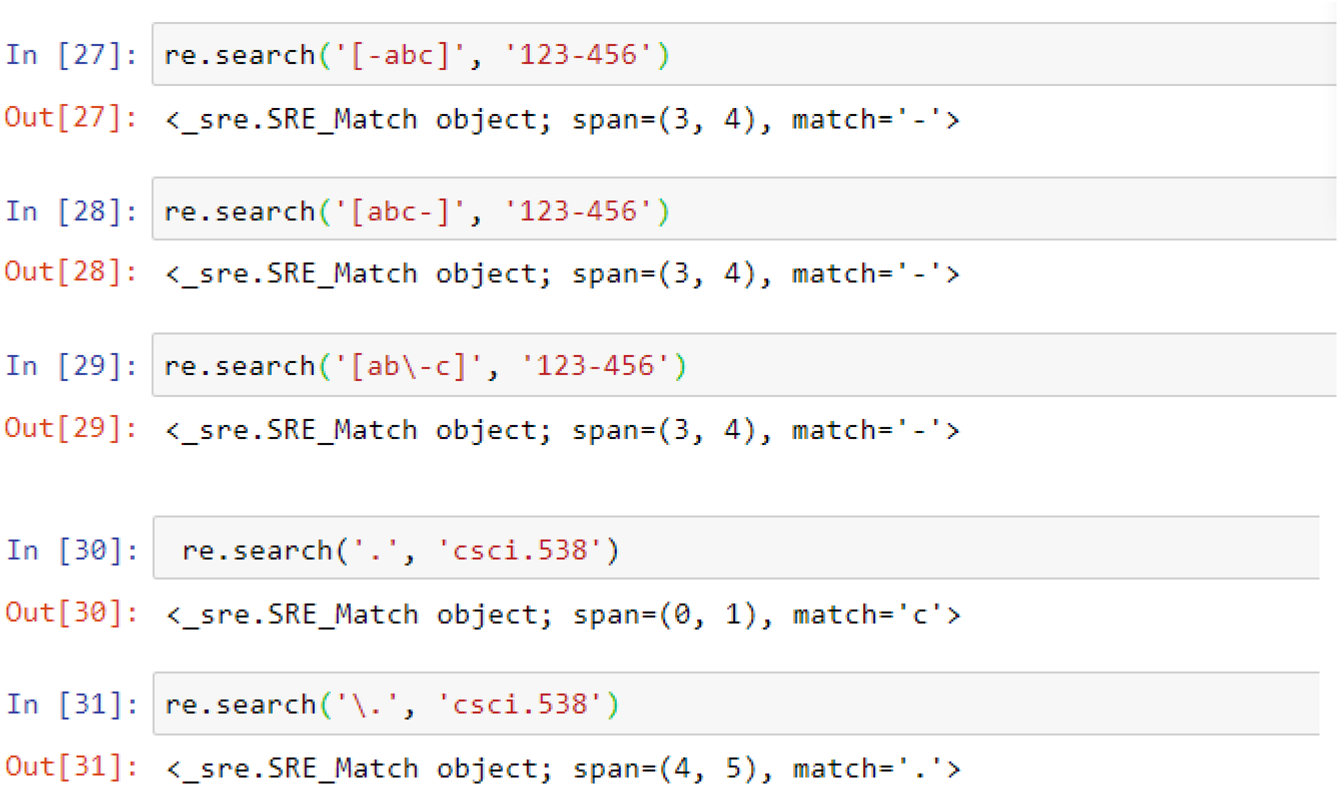
^

- Matches the beginning of a line or string.

```
string1 = "Hello World"
if re.search(r"^He", string1):
   print(string1, "starts with the
characters 'He’”)
```

Metacharacters - Escapes (\)

- place hyphen (-) as the first or last character or escape it with a backslash (\):



```
In [27]: re.search('[-abc]', '123-456')
Out[27]: <_sre.SRE_Match object; span=(3, 4), match='-'>

In [28]: re.search('[abc-]', '123-456')
Out[28]: <_sre.SRE_Match object; span=(3, 4), match='-'>

In [29]: re.search('[ab\-c]', '123-456')
Out[29]: <_sre.SRE_Match object; span=(3, 4), match='-'>

In [30]:  re.search('.', 'csci.538')
Out[30]: <_sre.SRE_Match object; span=(0, 1), match='c'>

In [31]: re.search('\.', 'csci.538')
Out[31]: <_sre.SRE_Match object; span=(4, 5), match='.'>
```

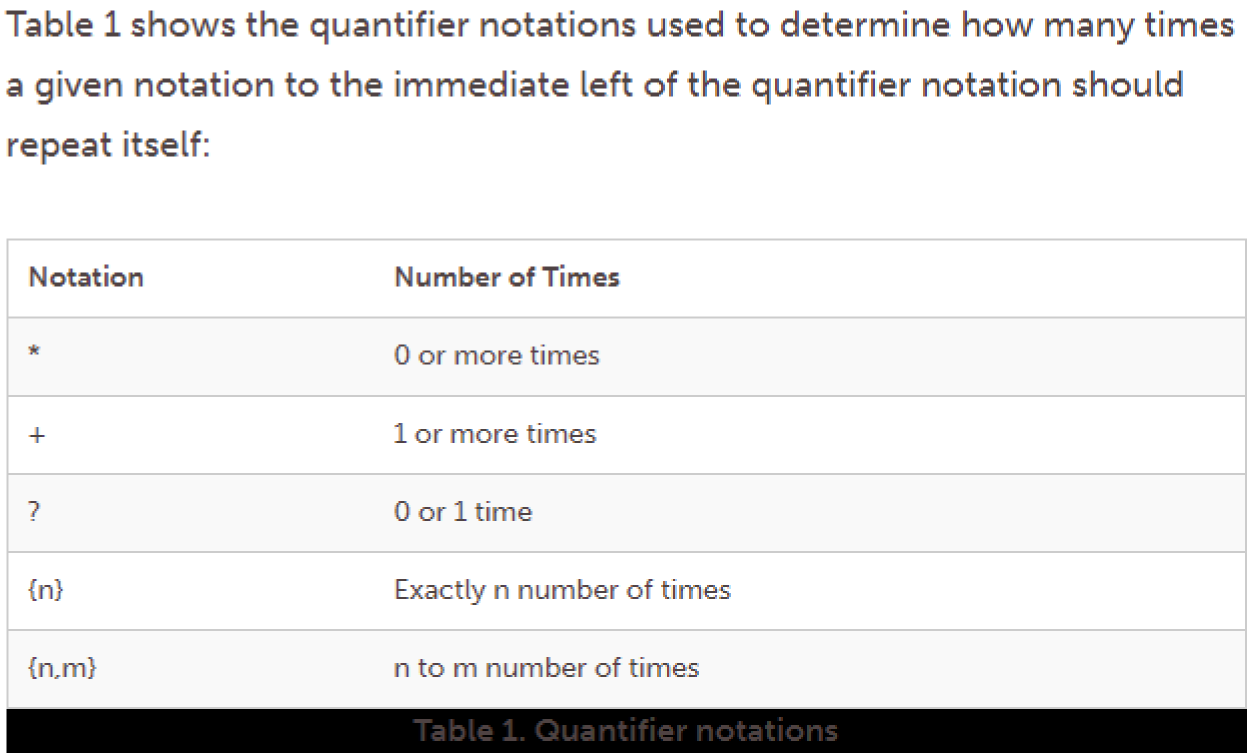


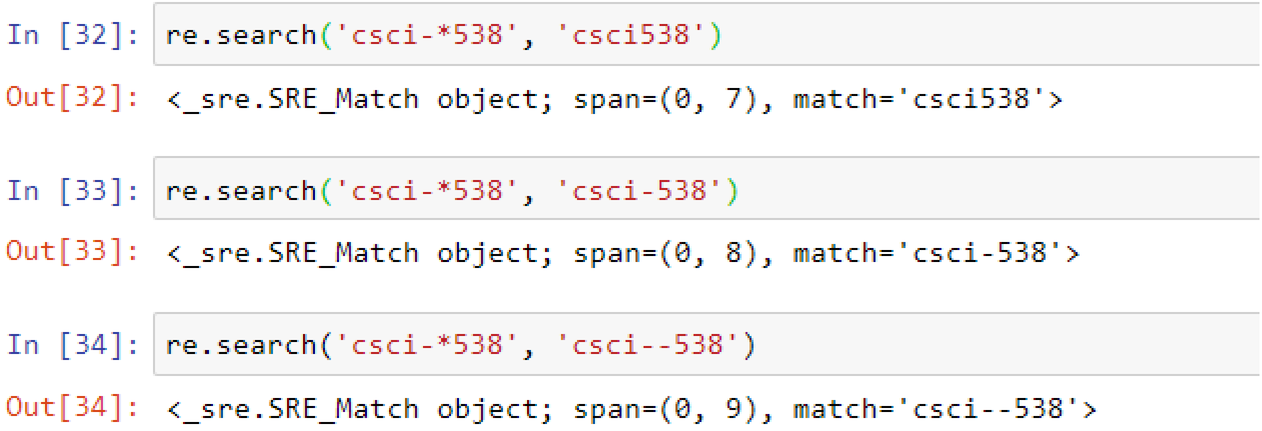
Table 1 shows the quantifier notations used to determine how many times a given notation to the immediate left of the quantifier notation should repeat itself:

Notation	Number of Times
*	0 or more times
+	1 or more times
?	0 or 1 time
{n}	Exactly n number of times
{n,m}	n to m number of times

Table 1. Quantifier notations

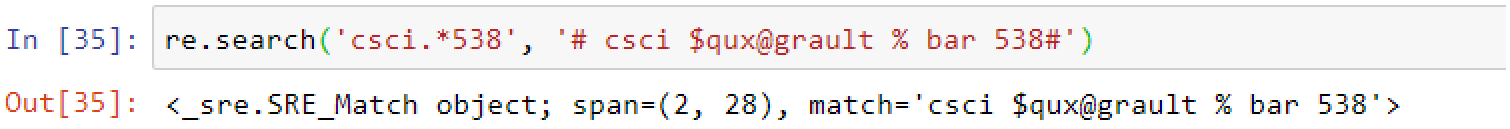
Metacharacters – Quantifiers (*,+,?)

- A quantifier metacharacter immediately follows a portion of a <regex> and indicates how many times that portion must occur for the match to succeed.
  - * : Matches zero or more repetitions of the preceding regex.
    - For example, a* matches zero or more 'a' characters. That means it would match an empty string, 'a', 'aa', 'aaa', and so on.



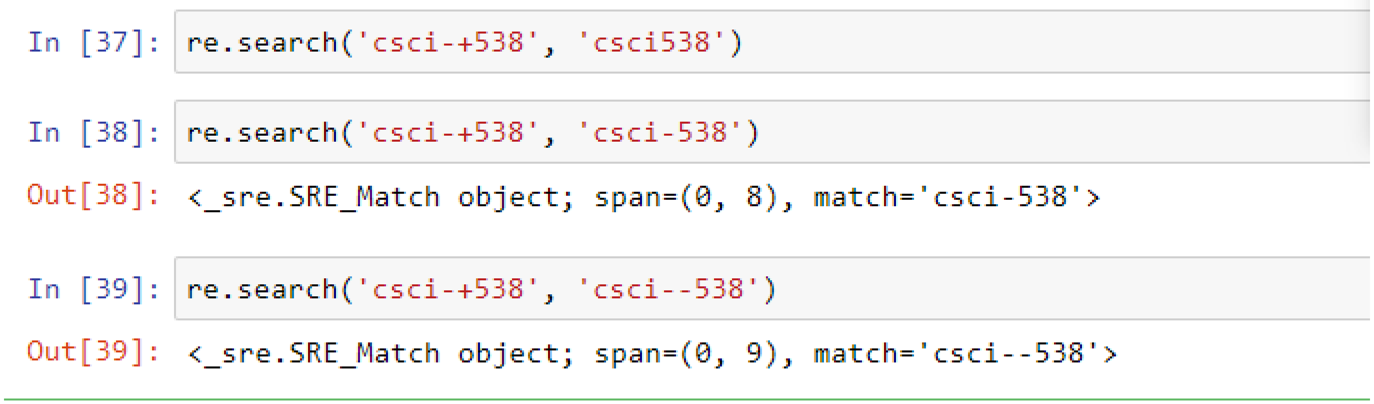
```
In [32]: re.search('csci-*538', 'csci538')
Out[32]: <_sre.SRE_Match object; span=(0, 7), match='csci538'>

In [33]: re.search('csci-*538', 'csci-538')
Out[33]: <_sre.SRE_Match object; span=(0, 8), match='csci-538'>

In [34]: re.search('csci-*538', 'csci--538')
Out[34]: <_sre.SRE_Match object; span=(0, 9), match='csci--538'>
```

- .* matches everything between ‘csci' and ‘538’:



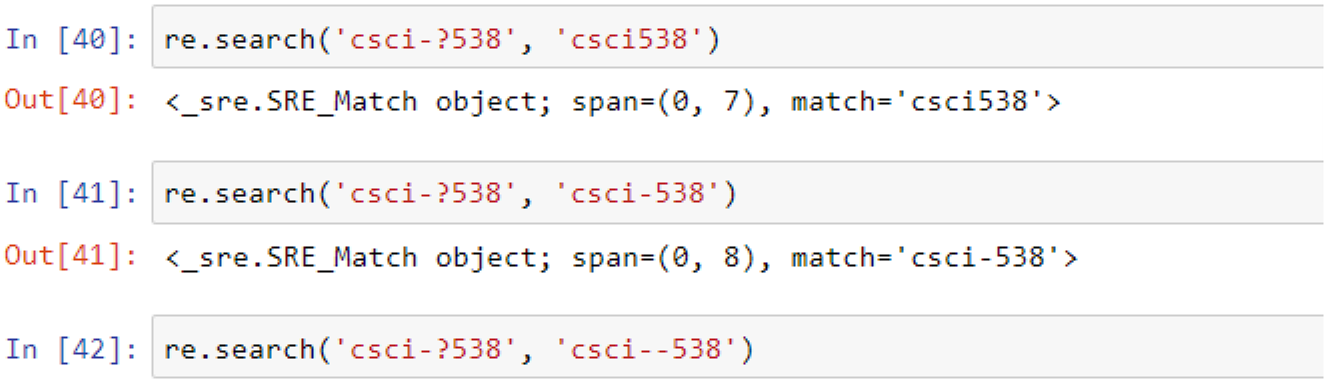
```
In [35]: re.search('csci.*538', '# csci $qux@grault % bar 538#')
Out[35]: <_sre.SRE_Match object; span=(2, 28), match='csci $qux@grault % bar 538'>
```

- +: Matches one or more repetitions of the preceding regex
    - This is similar to *, but the quantified regex must occur at least once:



```
In [37]: re.search('csci-+538', 'csci538')

In [38]: re.search('csci-+538', 'csci-538')
Out[38]: <_sre.SRE_Match object; span=(0, 8), match='csci-538'>

In [39]: re.search('csci-+538', 'csci--538')
Out[39]: <_sre.SRE_Match object; span=(0, 9), match='csci--538'>
```

- ?: Matches zero or one repetitions of the preceding regex



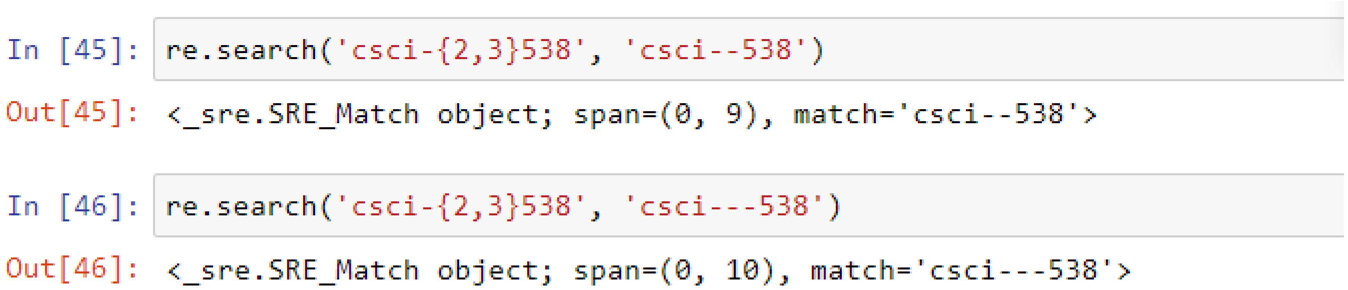
```
In [40]: re.search('csci-?538', 'csci538')
Out[40]: <_sre.SRE_Match object; span=(0, 7), match='csci538'>

In [41]: re.search('csci-?538', 'csci-538')
Out[41]: <_sre.SRE_Match object; span=(0, 8), match='csci-538'>

In [42]: re.search('csci-?538', 'csci--538')
```

  - In this example, there are matches on lines 40 and 41. But on line 42, where there are two '-' characters, the match fails.

- {m}: Matches exactly m repetitions of the preceding regex.

- {m,n}: Matches any number of repetitions of the preceding regex from m to n, inclusive.

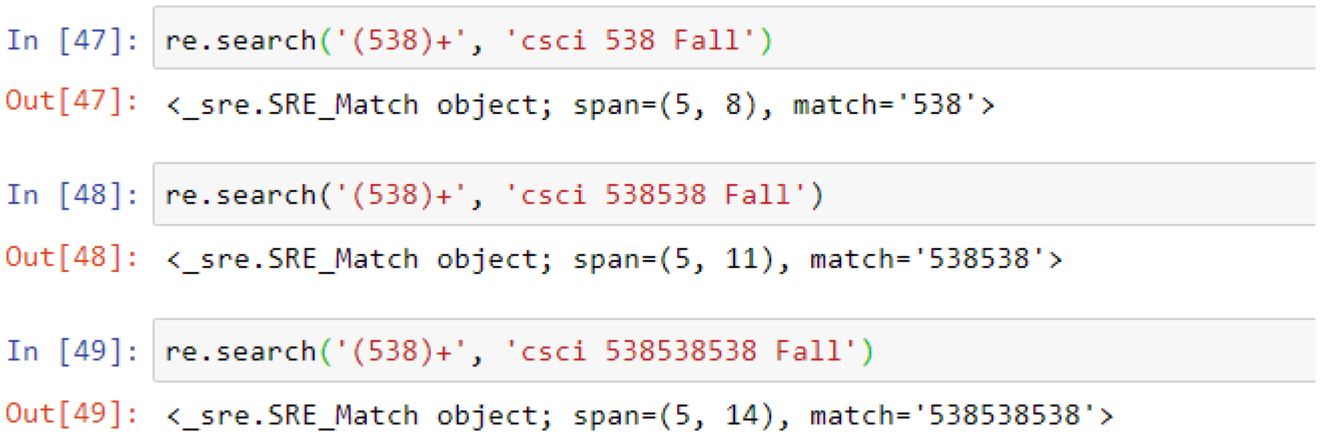
- {m}: Matches exactly m repetitions of the preceding regex.
- {m,n}: Matches any number of repetitions of the preceding regex from m to n, inclusive.



```
In [45]: re.search('csci-{2,3}538', 'csci--538')
Out[45]: <_sre.SRE_Match object; span=(0, 9), match='csci--538'>

In [46]: re.search('csci-{2,3}538', 'csci---538')
Out[46]: <_sre.SRE_Match object; span=(0, 10), match='csci---538'>
```

Grouping Constructs and Backreferences

- Grouping constructs break up a regex in Python into subexpressions or groups. This serves two purposes:
  - Grouping: A group represents a single syntactic entity. Additional metacharacters apply to the entire group as a unit.
  - Capturing: Some grouping constructs capture the portion of the search string that matches the subexpression in the group. You can retrieve captured matches later through several different mechanisms.
- (<regex>) : Defines a subexpression or group.

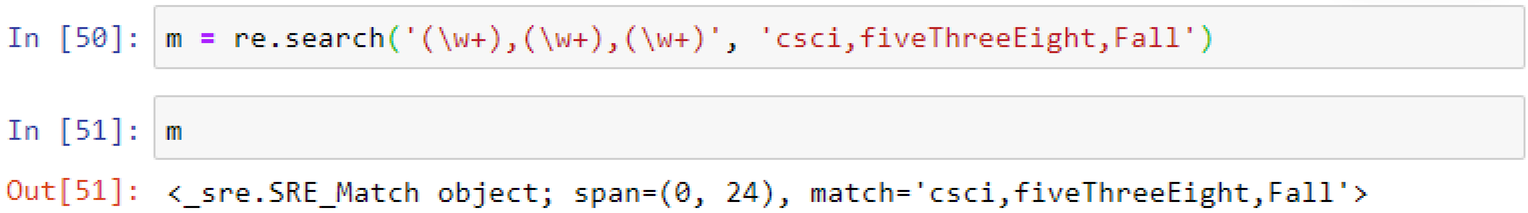


```
In [47]: re.search('(538)+', 'csci 538 Fall')
Out[47]: <_sre.SRE_Match object; span=(5, 8), match='538'>

In [48]: re.search('(538)+', 'csci 538538 Fall')
Out[48]: <_sre.SRE_Match object; span=(5, 11), match='538538'>

In [49]: re.search('(538)+', 'csci 538538538 Fall')
Out[49]: <_sre.SRE_Match object; span=(5, 14), match='538538538'>
```

- Difference between the two regexes with and without grouping parentheses:

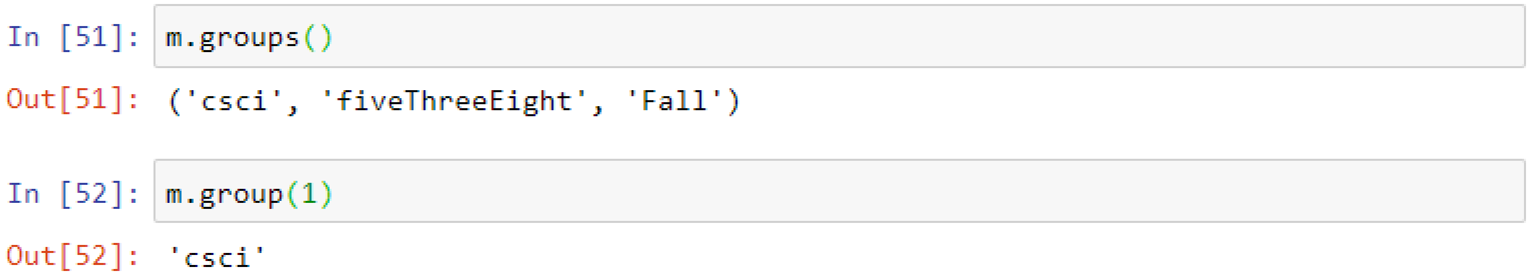
Regex	Interpretation	Matches	Examples
538+	The + metacharacter applies only to the character ’8'.	'53' followed by one or more occurrences of ’8'	'538’ ‘5388’ ‘53888'
(538)+	The + metacharacter applies to the entire string '538'.	One or more occurrences of '538'	'538' '538538' '538538538'

- There are two methods defined for a match object that provide access to captured groups: .groups() and .group().



```
In [50]: m = re.search('(\w+),(\w+),(\w+)', 'csci,fiveThreeEight,Fall')

In [51]: m

Out[51]: <_sre.SRE_Match object; span=(0, 24), match='csci,fiveThreeEight,Fall'>
```

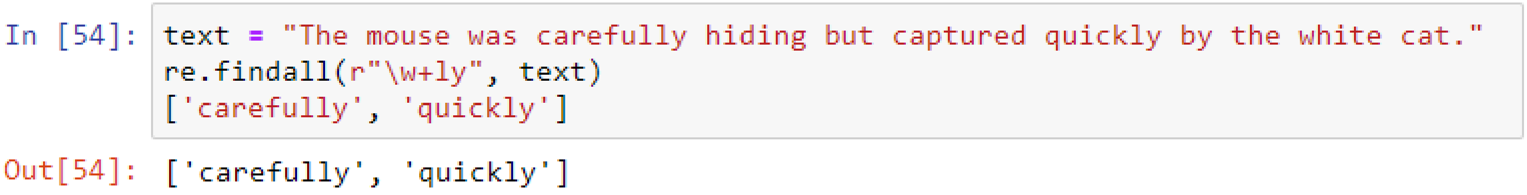
  - Each of the three (\w+) expressions matches a sequence of word characters. The full regex (\w+),(\w+),(\w+) breaks the search string into three comma-separated tokens.



```
In [51]: m.groups()

Out[51]: ('csci', 'fiveThreeEight', 'Fall')

In [52]: m.group(1)

Out[52]: 'csci'
```

re.findall (A,B)

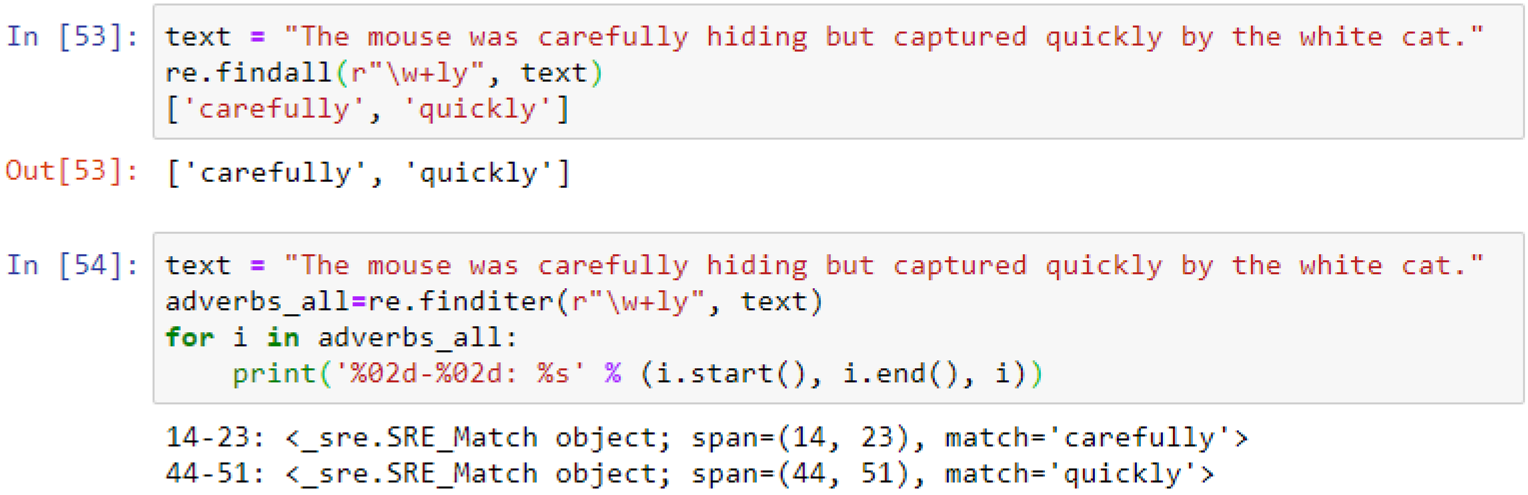
- re.findall(A,B) : Matches all instances of an expression A in a string B and returns them in a list, not just the first one as search() does.
  - For example, if we want to find all of the adverbs in some text, they might use findall() in the following manner:



```
In [54]: text = "The mouse was carefully hiding but captured quickly by the white cat."
         re.findall(r"\w+ly", text)
         ['carefully', 'quickly']

Out[54]: ['carefully', 'quickly']
```

  - Finding all Adverbs and their Position

- Finding all Adverbs and their Position



```
In [53]: text = "The mouse was carefully hiding but captured quickly by the white cat."
         re.findall(r"\w+ly", text)
         ['carefully', 'quickly']

Out[53]: ['carefully', 'quickly']

In [54]: text = "The mouse was carefully hiding but captured quickly by the white cat."
         adverbs_all=re.finditer(r"\w+ly", text)
         for i in adverbs_all:
             print('%02d-%02d: %s' % (i.start(), i.end(), i))

         14-23: <_sre.SRE_Match object; span=(14, 23), match='carefully'>
         44-51: <_sre.SRE_Match object; span=(44, 51), match='quickly'>
```

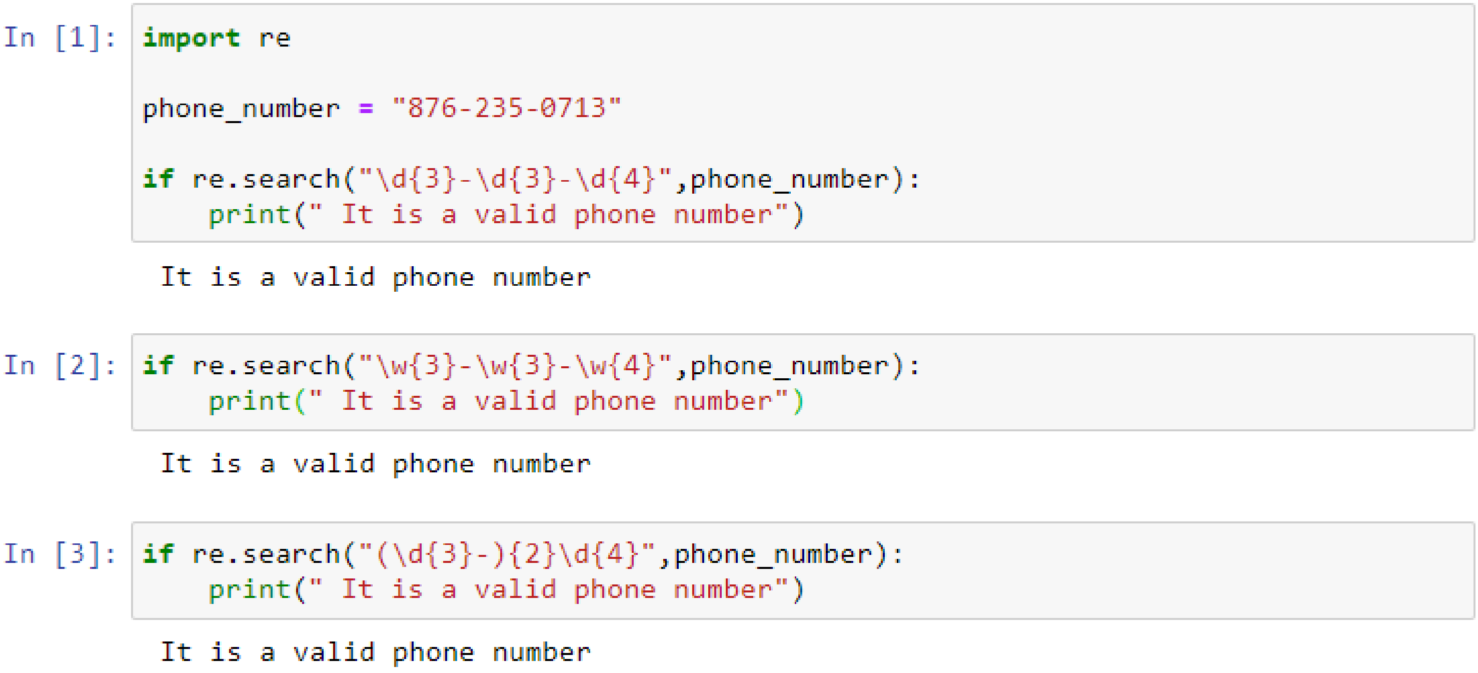
- If one wants more information about all matches of a pattern than the matched text, finditer() is useful as it provides match objects instead of strings

Applications: Phone Number Verification

444-123-2344
10-102904562
109-2937-034
100-2939-9390

- Problem statement:
  - To verify the phone numbers in US, we need to know that
    - 3 starting digits and ‘-’ sign
    - 3 middle digits and ‘-’ sign
    - 4 digits in the end

- Solution:



```
In [1]: import re

        phone_number = "876-235-0713"

        if re.search("\d{3}-\d{3}-\d{4}",phone_number):
            print(" It is a valid phone number")

         It is a valid phone number

In [2]: if re.search("\w{3}-\w{3}-\w{4}",phone_number):
            print(" It is a valid phone number")

         It is a valid phone number

In [3]: if re.search("(\d{3}-){2}\d{4}",phone_number):
            print(" It is a valid phone number")

         It is a valid phone number
```

\w : this is equivalent to [a-zA-Z0-9_]

\W : the equivalent of [^a-zA-Z0-9_]

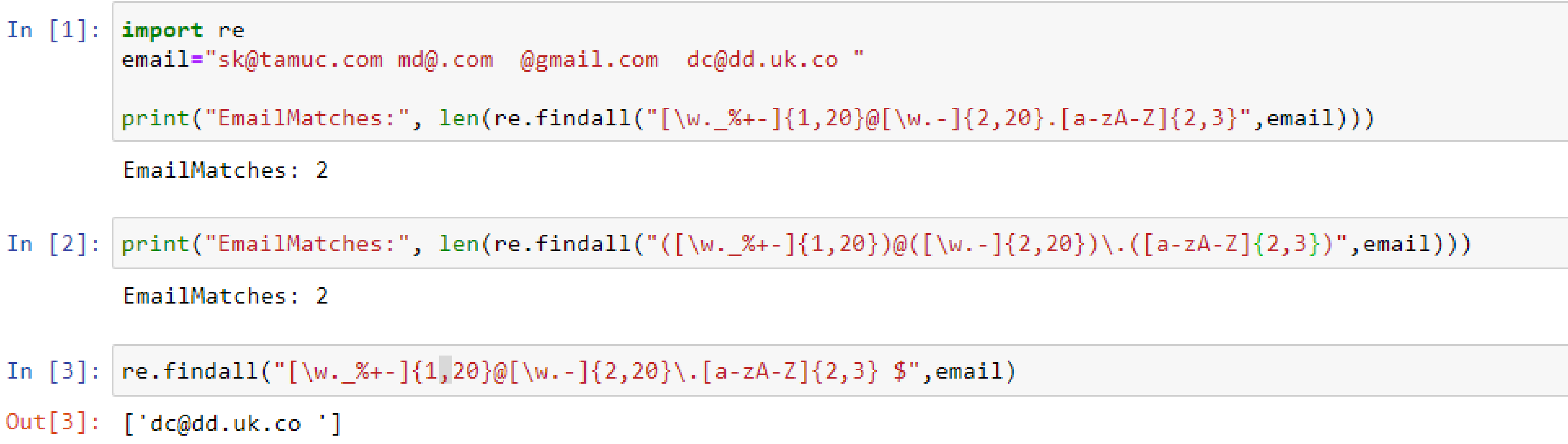
Character	Meaning
\d	Matches a digit in 0-9
\D	Matches a non-digit(anything but not a digit)
\w	Matches a word like a letter, digit, or underscore(_)
\W	Matches a non-word
\s	Matches a whitespace (space,\t,\r, or\n)
\S	Matches a non-whitespace

Application 2: E-mail Verification

- Problem statement:
  - To verify the phone numbers in US, we need to know that an E-mail address should include
    - 1-20 lowercase and uppercase letters, numbers, plus . _%+-
    - An @ symbol
    - 2-20 lowercase and uppercase letters, numbers, plus . –
    - A period
    - 2 to 3 lowercase and uppercase letters

Sc # .com
sk@gmail1.com
dk@gmail.com
s@k2@ga,cl.com
Cf # 4 @ c m

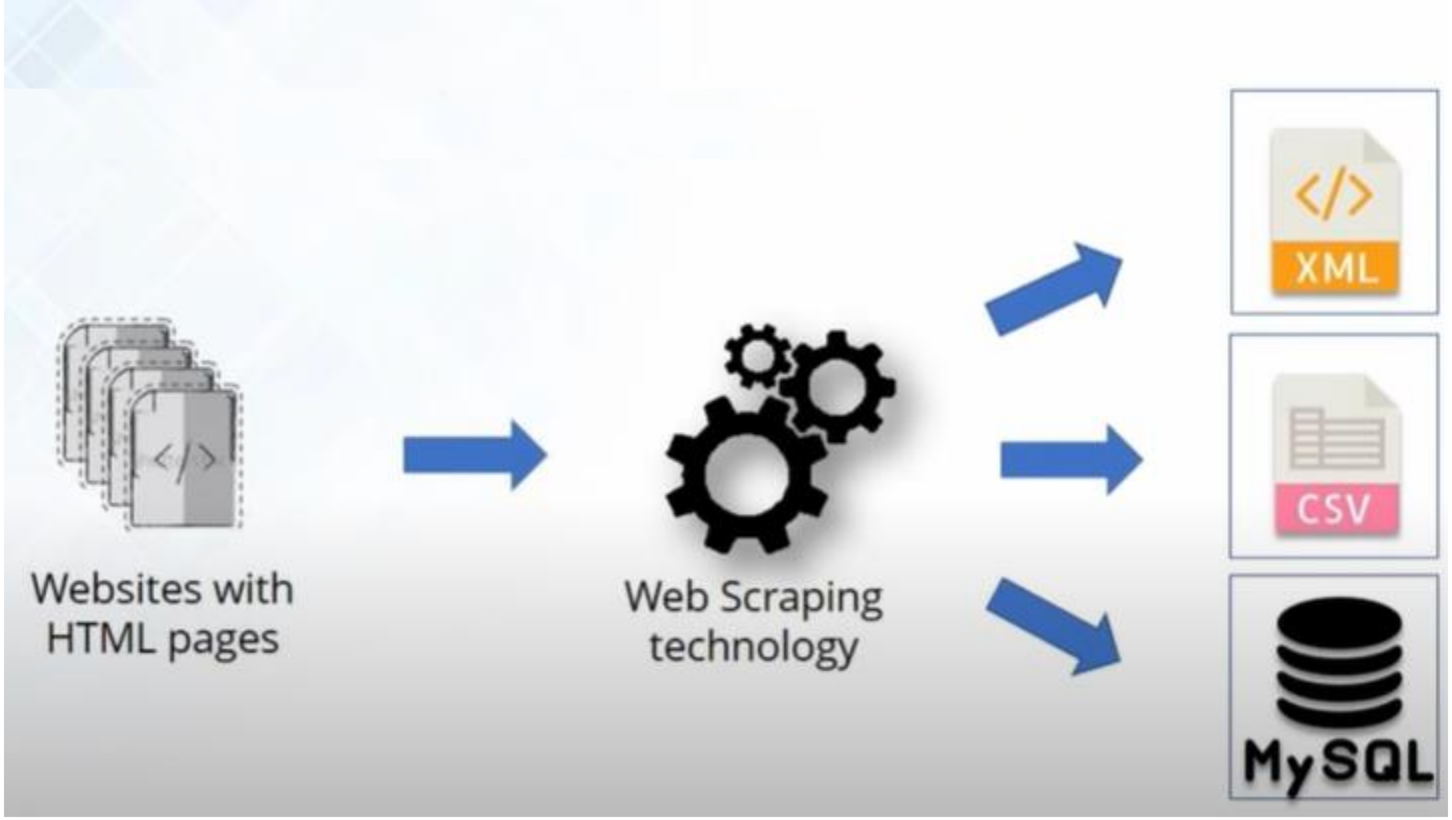
- Solution



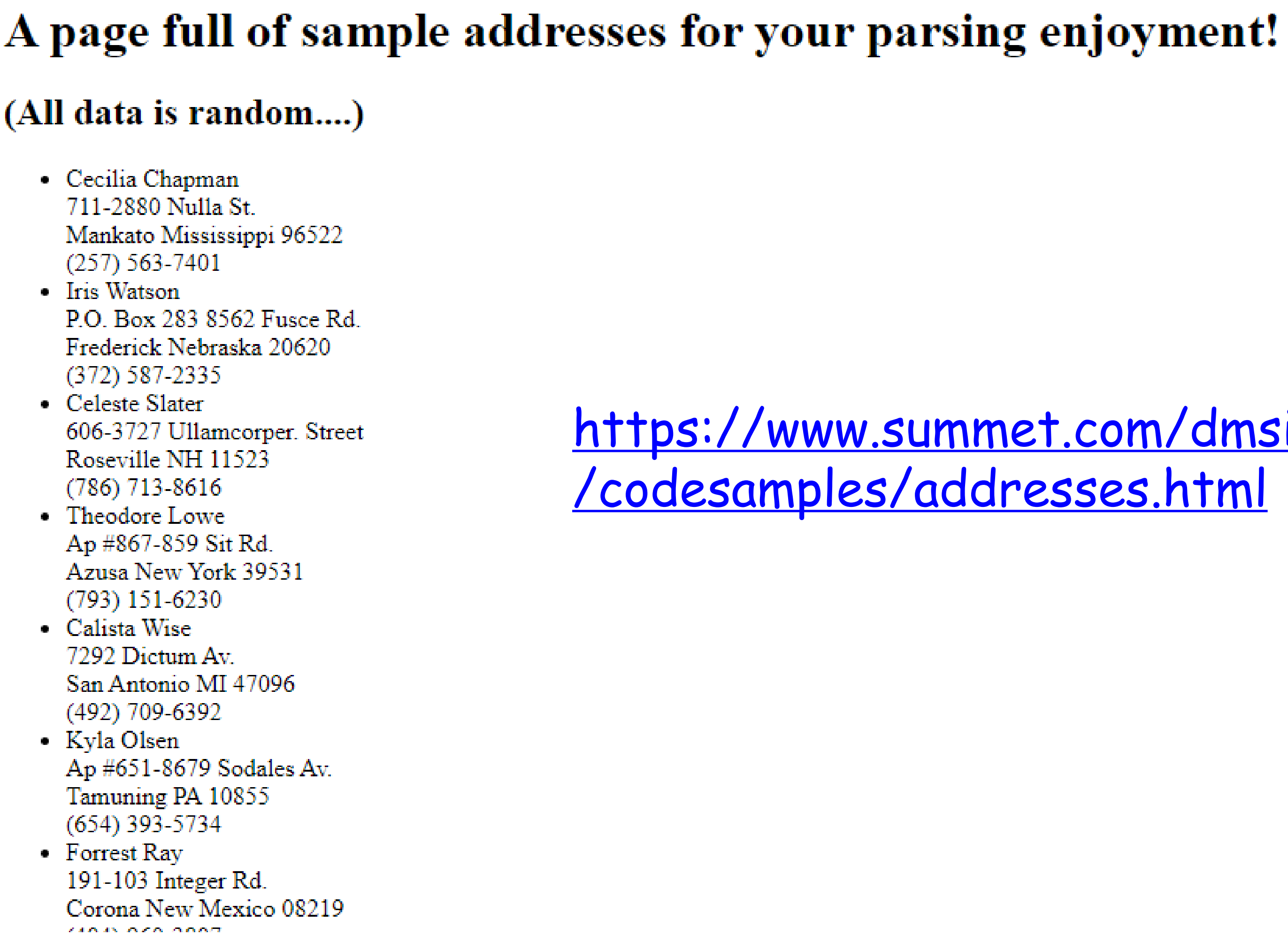
```
In [1]: import re
        email="sk@tamuc.com md@.com  @gmail.com  dc@dd.uk.co "

        print("EmailMatches:", len(re.findall("[\w._%+-]{1,20}@[\w.-]{2,20}.[a-zA-Z]{2,3}",email)))

        EmailMatches: 2

In [2]: print("EmailMatches:", len(re.findall("([\w._%+-]{1,20})@([\w.-]{2,20})\.([a-zA-Z]{2,3})",email)))

        EmailMatches: 2

In [3]: re.findall("[\w._%+-]{1,20}@[\w.-]{2,20}\.[a-zA-Z]{2,3} $",email)

Out[3]: ['dc@dd.uk.co ']
```

Application 3: Web scraping

- Problem statement:
  - Scrap all the phone numbers from a webpage using Regex



Visit the webpage



A page full of sample addresses for your parsing enjoyment!

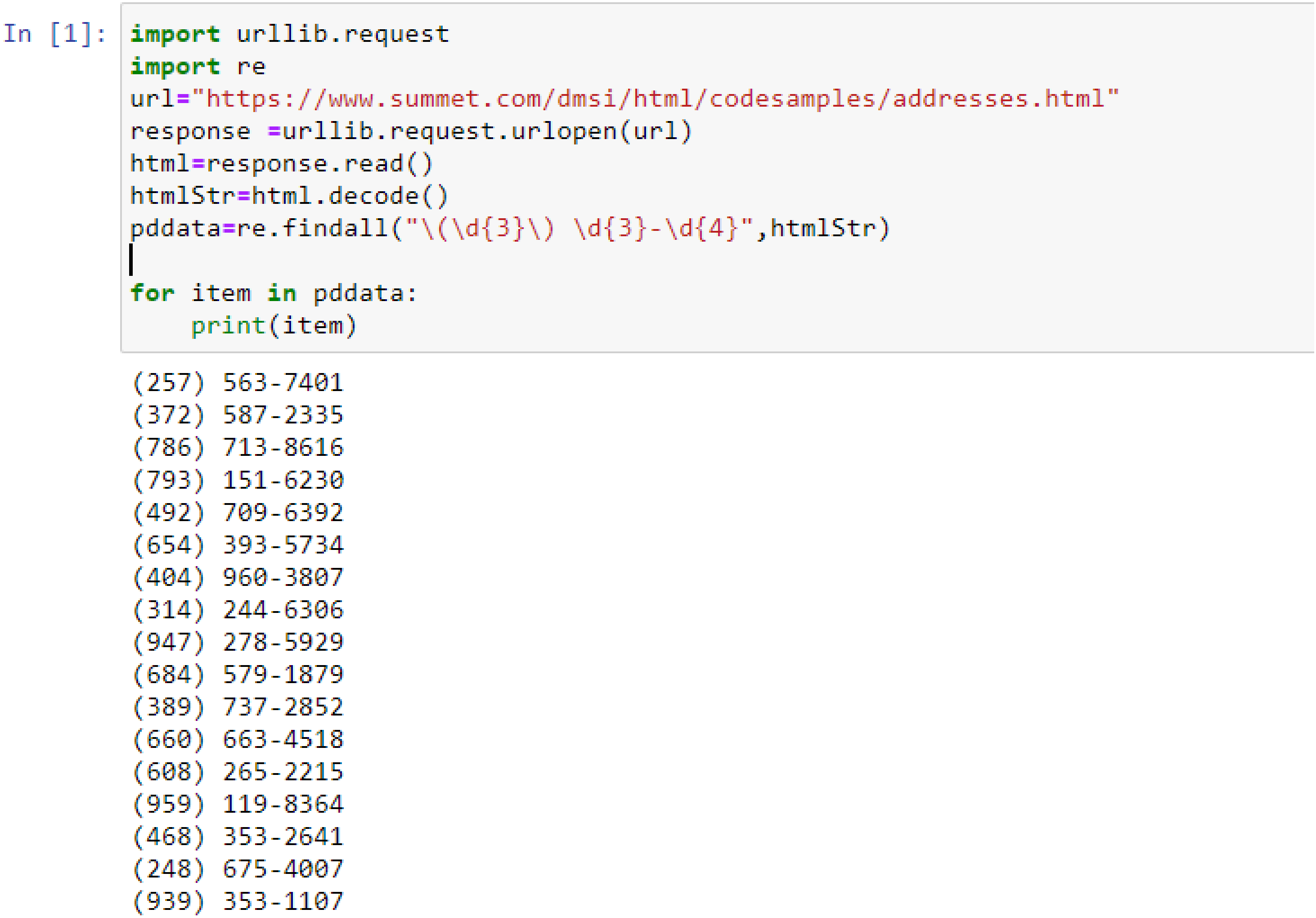
(All data is random....)

- Cecilia Chapman
  711-2880 Nulla St.
  Mankato Mississippi 96522
  (257) 563-7401
- Iris Watson
  P.O. Box 283 8562 Fusce Rd.
  Frederick Nebraska 20620
  (372) 587-2335
- Celeste Slater
  606-3727 Ullamcorper. Street
  Roseville NH 11523
  (786) 713-8616
- Theodore Lowe
  Ap #867-859 Sit Rd.
  Azusa New York 39531
  (793) 151-6230
- Calista Wise
  7292 Dictum Av.
  San Antonio MI 47096
  (492) 709-6392
- Kyla Olsen
  Ap #651-8679 Sodales Av.
  Tamuning PA 10855
  (654) 393-5734
- Forrest Ray
  191-103 Integer Rd.
  Corona New Mexico 08219

https://www.summet.com/dmsi/html/codesamples/addresses.html

- Solution:
  - pip install url

```
import urllib.request
import re
url="https://www.summet.com/dmsi/html/codesamples/addresses.html"
response =urllib.request.urlopen(url)
html=response.read()
htmlStr=html.decode()
pddata=re.findall("\(\d{3}\) \d{3}-\d{4}",htmlStr)

for item in pddata:
    print(item)
```



In [1]:

```python
import urllib.request
import re
url="https://www.summet.com/dmsi/html/codesamples/addresses.html"
response =urllib.request.urlopen(url)
html=response.read()
htmlStr=html.decode()
pddata=re.findall("\(\d{3}\) \d{3}-\d{4}",htmlStr)

for item in pddata:
    print(item)
```

```
(257) 563-7401
(372) 587-2335
(786) 713-8616
(793) 151-6230
(492) 709-6392
(654) 393-5734
(404) 960-3807
(314) 244-6306
(947) 278-5929
(684) 579-1879
(389) 737-2852
(660) 663-4518
(608) 265-2215
(959) 119-8364
(468) 353-2641
(248) 675-4007
(939) 353-1107
```

e.g. Username

- r”^[a-z0-9_-]{3,16}$”
- Starts and ends with 3-16 numbers, letters, underscores or hyphens
- Any lowercase letter (a-z), number (0-9), an underscore, or a hyphen.
- At least 3 to 16 characters.
- Matches E.g. my-us3r_n4m3
- but not th1s1swayt00_l0ngt0beausername

e.g. Password

- r“^[a-z0-9_-]{6,18}$”
  - ^…$ describes both the **start and the end of the line** using the special **^** (**hat**) and **$** (**dollar sign**) metacharacters
- Starts and ends with 6-18 letters, numbers, underscores, hyphens.
- Matches e.g. myp4ssw0rd but not mypa$$w0rd