Section 10

Regular Expressions

Mohammed Asir Shahid

2021-08-03

Contents

1 Regular Expression Basics

In this lesson, we will be working with pattern matching and regular expressions. Regular expressions allow you to specify a pattern of text to search for.

An example of a text pattern would be a phone number. 415-555-0000. In the US, this is the standard way of writing up phone numbers. If we had that same number but without the hyphens, we would not recognize it as a phone number.

```
def isPhoneNumber(text):
    if len(text) != 12:
        return False
    for i in range(0, 3):
        if not text[i].isdecimal():
            return False
    if text[3] != '-':
        return False
    for i in range(4, 7):
        if not text[i].isdecimal():
            return False
    if text[7] != '-':
        return False
    for i in range(8, 12):
        if not text[i].isdecimal():
            return False
    return True

print(isPhoneNumber("415-555-1234"))
print(isPhoneNumber("My Phone Number"))
```

```
True
```

```
False
```

That's a lot of code for a relatively simple task. If we want to find phone numbers in large strings, we'd need to write some more code.

```
def isPhoneNumber(text):
    if len(text) != 12:
        return False
    for i in range(0, 3):
        if not text[i].isdecimal():
            return False
    if text[3] != '-':
        return False
    for i in range(4, 7):
        if not text[i].isdecimal():
            return False
    if text[7] != '-':
        return False
    for i in range(8, 12):
        if not text[i].isdecimal():
            return False
    return True

print(isPhoneNumber("415-555-1234"))
print(isPhoneNumber("My Phone Number"))

message="Call me at 415-555-1011 tomorrow, or 415-555-9999 for my office line"

foundNuumber=False

for i in range(len(message)):
    chunk=message[i:i+12]
    if isPhoneNumber(chunk) == True:
        print("Phone number found")
        foundNumber=True
if not foundNumber:
    print("Could not find any phone numbers")
```

```
True
```

```
False
Phone number found
Phone number found
```

1.1 The re Module

We can write the previous code much faster using regular expressions.

```
import re

message="Call me at 415-555-1011 tomorrow, or 415-555-9999 for my office line"

phoneNumRegex=re.compile(r"\d\d\d-\d\d\d-\d\d\d\d")

mo=phoneNumRegex.search(message)

print(type(mo))
print(mo.group())

print(phoneNumRegex.findall(message))
```

```
<class 're.Match'>
415-555-1011
['415-555-1011', '415-555-9999']
```

2 Regex Groups and the Pipe Character

We can try some more of Python's more powerful pattern matching capabilities.

Let's say we want to seperate the area code from a phone number.

```
import re

phoneNumRegex = re.compile(r"\d\d\d-\d\d\d-\d\d\d\d")
mo=phoneNumRegex.search("My number is 415-555-4242")
print(mo.group())
```

```
415-555-4242
```

2.1 Groups

Let's say we only want the phone number or only the phone number portion of the number. We can do this sing parentheses to mark out groups in the string.

```
import re

phoneNumRegex = re.compile(r"(\d\d\d)-(\d\d\d-\d\d\d\d)")
mo=phoneNumRegex.search("My number is 415-555-4242")
print(mo.group())

print(mo.group(1))
print(mo.group(2))
```

```
415-555-4242
415
555-4242
```

The parentheses there can be useful syntax when we want to find specific parts of something. However, what can we do when we want to find literal parentheses? We would escape them using parentheses.

```
import re

phoneNumRegex = re.compile(r"\(\d\d\d\) \d\d\d-\d\d\d\d")
mo=phoneNumRegex.search("My number is (415) 555-4242")
print(mo.group())
```

```
(415) 555-4242
```

2.2 Pipe Character |

Pipes can be used to match one of several patterns as part of the regular expression.

Let's say we wanted to match any of the strings "Batman", "Batmobile", "Batcopter", or "Batbat"

```
import re

batRegex = re.compile(r"Bat(man|mobile|copter|bat)")
mo=batRegex.search("Batmobile lost a wheel.")
print(mo.group())
print(mo.group(1))
```

```
Batmobile
mobile
```

If the search method can't find the regular expression pattern, it will return None. In that case, we can risk running into errors.

3 Repetition in Regex Patterns and Greedy/Nongreedy Matching

How can we match a certain number of repetitions of a group? For example, one or more repitions, between 7 and 10 repitions, etc.

3.1 ? (zero or one)

This says match the preceding group either 0 or 1 times.

```
import re

#batRegex=re.compile(r"Batman|Batwoman")
batRegex=re.compile(r"Bat(wo)?man")

mo=batRegex.search("The Adventures of Batman")
print(mo.group())
```

```
mo=batRegex.search("The Adventures of Batwoman")
print(mo.group())

mo=batRegex.search("The Adventures of Batwowoman")
print(mo)
```

```
Batman
Batwoman
None
```

Using our earlier phone number example, we can make a regular expression that looks for phone numbers that do or do not have an area code. With our previous code, if we did not have an area code then the regex would not find the phone number.

```
import re

phoneRegex=re.compile(r"\d\d\d-\d\d\d-\d\d\d\d")

mo = phoneRegex.search("My phone number is 415-555-1234")
print(mo.group())

mo = phoneRegex.search("My phone number is 555-1234")
print(mo)

phoneRegex=re.compile(r"(\d\d\d-)?\d\d\d-\d\d\d\d")

mo = phoneRegex.search("My phone number is 415-555-1234")
print(mo.group())

mo = phoneRegex.search("My phone number is 555-1234")
print(mo.group())
```

```
415-555-1234
None
415-555-1234
555-1234
```

If we need to match a question mark as part of the expression, we can simply escape it by doing \?.

3.2 * (zero or more)

The asterisk means match 0 or more times.

```
import re
batRegex=re.compile(r"Bat(wo)*man")

mo=batRegex.search("The Adventures of Batman")
print(mo.group())

mo=batRegex.search("The Adventures of Batwoman")
print(mo.group())

mo=batRegex.search("The Adventures of Batwowowoman")
print(mo.group())
```

```
Batman
Batwoman
Batwowowoman
```

If you need to match an * that appears in the pattern, you can escape it by doing *.

3.3 + (one or more)

Unlike the star, the group preceding a + must appear in the pattern.

```
import re
batRegex=re.compile(r"Bat(wo)+man")

mo=batRegex.search("The Adventures of Batman")
print(mo)

mo=batRegex.search("The Adventures of Batwoman")
print(mo.group())
```

```
mo=batRegex.search("The Adventures of Batwowowoman")
print(mo.group())
```

```
None
Batwoman
Batwowowoman
```

If you need to match a + that appears in the pattern, you can escape it by doing \+.

3.4 Escaping ?, *, and +

```
import re
regex = re.compile(r"\+\*\?")

mo=regex.search("I learned about +*? regex syntax")
print(mo.group())
```

```
+*?
```

We could also put the above +*? into a group and then putting a + after it to say that the group needs to appear at least once.

```
import re
regex = re.compile(r"(\+\*\?)+")

mo=regex.search("I learned about +*?+*?+*? regex syntax")
print(mo.group())
```

```
+*?+*?+*?
```

3.5 {x} (exactly x)

This can be used if you wanted to match a specific number of repetitions of a group.

```
import re

haRegex=re.compile(r"(Ha){3}")
mo=haRegex.search("He said \"HaHaHa\"")
print(mo.group())
```

```
HaHaHa
```

While the above is a simple example, we could do it for many other, more complex examples.

```
import re

phoneRegex=re.compile(r"((\d\d\d-)?\d\d\d-\d\d\d\d(,)?){3}")
mo=phoneRegex.search("My numbers are 415-555-1234,555-4242,212-555-0000")
print(mo.group())
```

```
415-555-1234,555-4242,212-555-0000
```

3.6 {x,y} (at least x, at most y)

```
import re
haRegex=re.compile(r"(Ha){3,5}")
mo=haRegex.search("He said \"HaHaHa\"")
print(mo.group())

mo=haRegex.search("He said \"HaHaHaHaHa\"")
print(mo.group())

mo=haRegex.search("He said \"HaHaHaHaHaHa\"")
print(mo)
```

```
HaHaHa
HaHaHaHaHa
<re.Match object; span=(9, 19), match='HaHaHaHaHa'>
```

We can also have no y value which would have no maximum and be unbounded, x or more.

```
import re

digitRegex=re.compile(r"(\d){3,5}")
mo=digitRegex.search("1234567890")
print(mo.group())
```

```
12345
```

As we can see above, there was a match of the first 5 digits even though the first 3 also would have sufficed. By default, Python regular expressions do greedy matches. This means that it tries to match the longest possible string that matches the pattern.

In order to do a nongreedy match, we can specify a question mark following the curly braces. Then it matches the first, shortest pattern.

```
import re

digitRegex=re.compile(r"(\d){3,5}?")
mo=digitRegex.search("1234567890")
print(mo.group())
```

```
123
```

4 Regex Character Classes and the findall() Method

In this lesson, we'll talk about the findall() method with regular expressions.

```
import re

resume="""
JESSE KENDALL
123 Elm Street, Fall River, MA 02723, Cell: 508-555-5555, Home: 508-555-1234 twemel@cha
```

```
SUMMARY OF QUALIFICATIONS
Dedicated cell phone sales professional with demonstrated success in retail management

SALES SUCCESS
RETAIL STORE SALES MANAGER, 20xx - 20xx
ABC CELLULAR, Fall River, MA
Recruited, hired, trained, developed, and directed retail sales teams for two retail AB

Implemented a sales-tracking spreadsheet to replace a manual form writing process to in
Developed innovative and effective marketing programs; exceeded store sales quotas.
Successfully managed one of the highest-producing ABC Wireless dealer locations in the
Received several ‘‘Sales Manager of the Month’’ Awards.
Created a team spirit within the stores that resulted in increased sales, long-term emp
CELL PHONE SALES REPRESENTATIVE, 20xx - 20xx
BCD CELL PHONE HUT, Fall River, MA
Partnered with a high-performing sales staff to provided quality customer service. Serv

Effectively delivered post-sale care services, exceeding clients’ expectations in a cos
Obtained significant business by delivering presale presentations to demonstrate new ph
CUSTOMER SERVICE REPRESENTATIVE, 20xx - 20xx
CDE CELLULAR SERVICES, Fall River, MA
Responded to billing inquiries, assisted in technical troubleshooting, and performed ra

Chosen to facilitate training in an outsourced call center.
EDUCATION
Bachelor of Arts in Communication (Major: Advertising), 20xx
XYZ UNIVERSITY, Milwaukee, WI

REFERENCES
Excellent references provided upon request.
"""


phoneRegex=re.compile(r"\d{3}-\d{3}-\d{4}")
mo=phoneRegex.findall(resume)
print(mo)
```

```
['508-555-5555', '508-555-1234']
```

We used findall() above so we can easily find all of the matches. If we used search() like we did earlier, it would only return the first match.

There is an important distinction between search() and findall(). The search() method returns Match Objects while the findall() method returns a list of strings.

If the regular expression string has 0 or 1 groups, then the findall() method will jstu return a list of strings which each string in the list is the text that was found.

However, with regex objects that have 2 or more groups, the following occurs.

```
import re

resume="""
JESSE KENDALL
123 Elm Street, Fall River, MA 02723, Cell: 508-555-5555, Home: 508-555-1234 twemel@cha



SUMMARY OF QUALIFICATIONS
Dedicated cell phone sales professional with demonstrated success in retail management

SALES SUCCESS
RETAIL STORE SALES MANAGER, 20xx - 20xx
ABC CELLULAR, Fall River, MA
Recruited, hired, trained, developed, and directed retail sales teams for two retail A

Implemented a sales-tracking spreadsheet to replace a manual form writing process to i
Developed innovative and effective marketing programs; exceeded store sales quotas.
Successfully managed one of the highest-producing ABC Wireless dealer locations in the
Received several ‘‘Sales Manager of the Month’’ Awards.
Created a team spirit within the stores that resulted in increased sales, long-term em
CELL PHONE SALES REPRESENTATIVE, 20xx - 20xx
BCD CELL PHONE HUT, Fall River, MA
Partnered with a high-performing sales staff to provided quality customer service. Ser

Effectively delivered post-sale care services, exceeding clients’ expectations in a co
```

```
Obtained significant business by delivering presale presentations to demonstrate new p
CUSTOMER SERVICE REPRESENTATIVE, 20xx - 20xx
CDE CELLULAR SERVICES, Fall River, MA
Responded to billing inquiries, assisted in technical troubleshooting, and performed ra

Chosen to facilitate training in an outsourced call center.
EDUCATION
Bachelor of Arts in Communication (Major: Advertising), 20xx
XYZ UNIVERSITY, Milwaukee, WI

REFERENCES
Excellent references provided upon request.
"""


phoneRegex=re.compile(r"(\d{3})-(\d{3}-\d{4})")
mo=phoneRegex.findall(resume)
print(mo)
```

```
[('508', '555-5555'), ('508', '555-1234')]
```

Now, instead of returning a list of strings, a list of tuples containing strings is returned. Each string in the tuple is the contents of each group.

4.1 Character Classes

We've already looked at one character class (). It represents any numeric digit between 0 and 9. Character classes are shortcuts that make our code easier to read and write.

Shorthand character class	Represents
. Any numeric digit from 0 to 9.	
	Any character that is not a numeric digit from 0 to 9.
	Any letter, numeric digit, or the underscore character. (Think of th
	Any character that is not a letter, numeric digit, or the underscore
	Any space, tab, or newline character. (Think of this as matching "s
§	Any character that is not a space, tab, or newline.

4.2 12 Days of Christmas Example

We can use regular expressions to find patterns where we have some number followed by some words.

```
import re

lyrics= "12 drummers drumming, 11 pipers piping, 10 lords a leaping, 9 ladies dancing,

xmasRegex=re.compile(r"\d+\s\w+")

print(xmasRegex.findall(lyrics))


['12 drummers', '11 pipers', '10 lords', '9 ladies', '7 swans', '6 geese', '5 golden',
```

4.3 Making Your Own Character Classes

Above we showed the existing shorthand character classes, but we can create our own character classes as well. Let's say we want to create a shorthand for vowels, then we can use r"[aeiouAEIOU]" for our Regex. If we want to make a negative character class, we can put in a ^ at the start of the square brackets.

```
import re
vowelRegex=re.compile(r"[aeiouAEIOU]")

print(vowelRegex.findall("Robocop eats baby food."))

doublevowelRegex=re.compile(r"[aeiouAEIOU]{2}")

print(doublevowelRegex.findall("Robocop eats baby food."))

consonantsRegex=re.compile(r"[^aeiouAEIOU]")

print(consonantsRegex.findall("Robocop eats baby food."))

['o', 'o', 'o', 'e', 'a', 'a', 'o', 'o']
['ea', 'oo']
['R', 'b', 'c', 'p', ' ', 't', 's', ' ', 'b', 'b', 'y', ' ', 'f', 'd', '.']
```

5 Regex Dot-Star and the Caret/Dollar Characters

In our last lesson, we learned that we can use a ˆ at the start of the square brackets in order to create a negative character class. We can also use a carrot at the start of a regular expression to indicate that the match needs to occur at the beginning of the text. We can also put a dollar sign at the end of the regular expression to indicate that the string has to match at the end with the regex pattern.

```
import re

beginsWithHelloRegex=re.compile(r"^Hello")

mo=beginsWithHelloRegex.search("Hello there")
print(mo.group())

mo=beginsWithHelloRegex.search("He said Hello!")
print(mo)

endsWithWorldRegex=re.compile(r"world!$")

mo=endsWithWorldRegex.search("Hello world!")
print(mo.group())

mo=endsWithWorldRegex.search("Hello world!!!")
print(mo)
```

```
Hello
None
world!
None
```

If we use both ˆ at the beginning and $ at the end of our regex then that must indicate that the pattern must match the entire string.

```
import re

allDigitsRegex= re.compile(r"^\d+$")
```

```
mo=allDigitsRegex.search("25168168416846516181 6")
print(mo.group())

mo=allDigitsRegex.search("25168168x4168465161816")
print(mo)
```

```
251681684168465161816
None
```

5.1 . (anything except newline)

Having a period in your regex stands for any character except for newline.

```
import re

atRegex=re.compile(r".at")
mo=atRegex.findall("The cat in the hat sat on the flat mat")
print(mo)
```

```
['cat', 'hat', 'sat', 'lat', 'mat']
```

We can see above that the regex did not match "flat" since the . character is only looking for a single character before the "at". That's why we instaed have "lat" get matched.

```
import re

atRegex=re.compile(r".{1,2}at")
mo=atRegex.findall("The cat in the hat sat on the flat mat")
print(mo)
```

```
[' cat', ' hat', ' sat', 'flat', ' mat']
```

5.2 Dot-Star to Match Anything

A common thing that is done is a .* pattern to match anything, any pattern whatsoever.

```
import re


nameRegex=re.compile(r"First Name: (.*) Last Name: (.*)")

mo=nameRegex.findall("First Name: Al Last Name: Sweigart")

print(mo)
```

```
[('Al', 'Sweigart')]
```

5.3 (.*) is greedy, (.*?) is non-greedy

By default, dot-star uses greedy mode. We need to add the ? in order to make it not greedy.

```
import re

serve = "<To serve humans> for dinner.>"

nongreedy=re.compile(r"<(.*?)>")

print(nongreedy.findall(serve))

greedy=re.compile(r"<(.*)>")

print(greedy.findall(serve))
```

```
['To serve humans']
['To serve humans> for dinner.']
```

5.4 Making Dot Match Newlines Too (with re.DOTALL)

Earlier, we mentioned that the dot character matches everything except for the newline character. How can we change that?

```
import re

prime="Serve the public trust. \nProtect the innocent. \nUphold the law."
print(prime)

dotStar=re.compile(r".*")

print(dotStar.search(prime))

dotStar=re.compile(r".*", re.DOTALL)

print(dotStar.search(prime))
```

```
Serve the public trust.
Protect the innocent.
Uphold the law.
<re.Match object; span=(0, 24), match='Serve the public trust. '>
<re.Match object; span=(0, 63), match='Serve the public trust. \nProtect the innocent.
```

5.5 re.IGNORECASE

Have second arguments to the compile function can be pretty useful. We can also have it do a case insensitive regex match.

```
import re

vowelRegex=re.compile(r"[aeiou]")
print(vowelRegex.findall("Al, why does your programming book talk about RoboCop so muc

vowelRegex=re.compile(r"[aeiou]", re.I)
print(vowelRegex.findall("Al, why does your programming book talk about RoboCop so muc
['o', 'e', 'o', 'u', 'o', 'a', 'i', 'o', 'o', 'a', 'a', 'o', 'u', 'o', 'o', 'o', 'o',
['A', 'o', 'e', 'o', 'u', 'o', 'a', 'i', 'o', 'o', 'a', 'a', 'o', 'u', 'o', 'o', 'o',
```

6 Regex sub() Method and Verbose Mode

Previously we called the re.compile function to create regular expression objects. These objects had search and findall methods. There are like the search features in word processors.

6.1 The sub() method

The sub() method is like the replace feature in word processors. Let's see how to use it.

```
import re

namesRegex=re.compile(r"Agent \w+")
mo=namesRegex.findall("Agent Alice gave the secret documents to Agent Bob.")
print(mo)

mo=namesRegex.sub("REDACTED","Agent Alice gave the secret documents to Agent Bob.")
print(mo)
```

```
['Agent Alice', 'Agent Bob']
REDACTED gave the secret documents to REDACTED.
```

What if instead of doing REDACTED, we want to change the name to Agent plus first initial?

We have to use a group and then call the group from the original text using \1.

```
import re

namesRegex=re.compile(r"Agent (\w)\w*")
mo=namesRegex.findall("Agent Alice gave the secret documents to Agent Bob.")
print(mo)
```

```
mo=namesRegex.sub(r"Agent \1******","Agent Alice gave the secret documents to Agent Bo
print(mo)
```

```
['A', 'B']
Agent A****** gave the secret documents to Agent B******.
```

6.2 Verbose Mode with re.VERBOSE

Regex strings can look awkward and difficult to parse, particularly when they get too long. We can use the verbose format in order to fix this.

```
import re

re.compile(r"""
(\d\d\d)|   # area code without parentheses, with dash
(\(\d\d\d\)) or area code with parentheses
-        # first dash
\d\d\d   # first 3 digits
-        # second dash
\d\d\d\d # last 4 digits
\sx\d{2,4} # extension like x1234""", re.VERBOSE)
```

6.3 Using Multiple Options (re.I, re.DOTALL, re.VERBOSE)

We have learned about 3 different options for our re.compile function. What if we want to use several of these for the same regex? We can combine them using pipe symbols.

```
import re

re.compile(r"""
(\d\d\d)|   # area code without parentheses, with dash
(\(\d\d\d\)) or area code with parentheses
-        # first dash
\d\d\d   # first 3 digits
-        # second dash
\d\d\d\d # last 4 digits
\sx\d{2,4} # extension like x1234""", re.VERBOSE | re.DOTALL | re.I)
```

7 Regex Example Program: A Phone and Email Scraper

Using what we have learned, we will create a scraper that takes out all the emails and phone numbers from a PDF.

We can create the phoneAndEmail.py script for this.