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
```