0 Introduction

Often you need to write the code to validate user input such as to check whether the input is a number, a string with all lowercase letters, or a social security number. How do you write this type of code? A simple and effective way to accomplish this task is to use the regular expression.

A regular expression (abbreviated regex) is a string that describes a pattern for matching a set of strings. Regular expression is a powerful tool for string manipulations. You can use regular expressions for matching, replacing, and splitting strings.

1 Getting Started

To use regex, import the `re` module. You can use the `split` function in the module to split a string. For example,

```
re.split(" ", "ab bc cd")
```

splits "ab bc cd" into a list ['ab', 'bc', 'cd'].

At first glance, `re.split` function is very similar to the `split` method in the string object. For example, you can use the following method to split "ab bc cd".

```
"ab bc cd".split()
```

However, the `re.split` function is more powerful. You can specify `regex` pattern to split a string. For example,

```
re.split("\d", "ab1bc4cd")
```

splits "ab1bc4cd" into a list ['ab', 'bc', 'cd']. \d in the preceding statement is a regular expression. It represents any single digit. Here is another example,

```
re.split("\d*", "ab13bc44cd443gg")
```

splits "ab13bc44cd443gg" into a list ['ab', 'bc', 'cd', 'gg']. Here, the regular expression \d* means zero or more digits.

2 Regular Expression Syntax

A regular expression consists of literal characters and special symbols. Table 1 lists some frequently used syntax for regular expressions.
### Table 1: Frequently Used Regular Expressions

<table>
<thead>
<tr>
<th>Regular Expression</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>A character literal</td>
<td>&quot;good&quot; matches &quot;good&quot;</td>
</tr>
<tr>
<td>.</td>
<td>Any single character</td>
<td>&quot;good&quot; matches &quot;goo.&quot;</td>
</tr>
<tr>
<td>(ab</td>
<td>cd)</td>
<td>ab or cd</td>
</tr>
<tr>
<td>[abc]</td>
<td>a, b, or c</td>
<td>&quot;good&quot; matches &quot;[ag]&quot;</td>
</tr>
<tr>
<td>(^abc)</td>
<td>any character except a, b, or c</td>
<td>&quot;good&quot; matches &quot;[^ac]&quot;</td>
</tr>
<tr>
<td>[a-z]</td>
<td>a through z</td>
<td>&quot;good&quot; matches [a-i]oo[a-d]</td>
</tr>
<tr>
<td>(^a-z)</td>
<td>any character except a through z</td>
<td>&quot;good&quot; matches goo[^i-x]</td>
</tr>
<tr>
<td>\d</td>
<td>a digit, same as [0-9]</td>
<td>&quot;good3&quot; matches &quot;good\d&quot;</td>
</tr>
<tr>
<td>\D</td>
<td>a non-digit</td>
<td>&quot;good&quot; matches &quot;\D\Dod&quot;</td>
</tr>
<tr>
<td>\w</td>
<td>a word character</td>
<td>&quot;good3&quot; matches &quot;goo\w\w&quot;</td>
</tr>
<tr>
<td>\W</td>
<td>a non-word character</td>
<td>$good matches &quot;\Wgood&quot;</td>
</tr>
<tr>
<td>\s</td>
<td>a whitespace character</td>
<td>&quot;good 2&quot; matches &quot;good\s2&quot;</td>
</tr>
<tr>
<td>\S</td>
<td>a non-whitespace character</td>
<td>&quot;good&quot; matches \Sood</td>
</tr>
<tr>
<td>p*</td>
<td>zero or more occurrences of pattern p</td>
<td>&quot;good&quot; matches &quot;a*&quot;</td>
</tr>
<tr>
<td>p+</td>
<td>one or more occurrences of pattern p</td>
<td>bbb matches &quot;a+&quot;</td>
</tr>
<tr>
<td>p?</td>
<td>zero or one occurrence of pattern p</td>
<td>&quot;good&quot; matches &quot;good?&quot;</td>
</tr>
<tr>
<td>p{n}</td>
<td>exactly n occurrences of pattern p</td>
<td>bbb matches &quot;b{n}&quot;</td>
</tr>
<tr>
<td>p{n,}</td>
<td>at least n occurrences of pattern p</td>
<td>aaa matches &quot;a{3}&quot;</td>
</tr>
<tr>
<td>p{n,m}</td>
<td>between n and m occurrences (inclusive)</td>
<td>good does not match &quot;go{2}d&quot;</td>
</tr>
</tbody>
</table>

**Note**
Recall that a whitespace (or a whitespace character) is any character which does not display itself but does take up space. The characters '\ ', '\t', '\n', '\r', '\f' are whitespace characters. So \s is the same as ['t'n]'r'f', and \S is the same as ['^ \t
\r\f\v'].

**Note**
A word character is any letter, digit, or the underscore character. So \w is the same as ['zZ][0-9]_'] or simply ['zZ-A-Z0-9_'], and \W is the same as [^[a-zA-Z0-9_]].

**Note**
The last six entries *, +, ?, {n}, {n,}, and {n, m} in Table 1 are called quantifiers that specify how many times the pattern before a quantifier may repeat. For example, A* matches zero or more A’s, A+ matches one or more A’s, A? matches zero or one A’s, A{3} matches exactly AAA, A{3,} matches at least three A’s, and A{3,6} matches between 3 and 6 A’s. *
is the same as \{0,\}, + is the same as \{1,\}, and ? is the same as \{0,1\}.

**CAUTION**
Do not use spaces in the repeat quantifiers. For example, A\{3,6\} cannot be written as A\{3,\ 6\} with a space after the comma.

**NOTE**
You may use parentheses to group patterns. For example, (ab)\{3\} matches ababab, but ab\{3\} matches abbb.

Let us use several examples to demonstrate how to construct regular expressions.

**Example 1:** The pattern for social security numbers is xxx-xx-xxxx, where x is a digit. A regular expression for social security numbers can be described as
\d\{3\}-\d\{2\}-\d\{4\}

For example,

"111-22-3333" matches "\d\{3\}-\d\{2\}-\d\{4\}"

but

"11-22-3333" does not match "\d\{3\}-\d\{2\}-\d\{4\}".

**Example 2:** An even number ends with digits 0, 2, 4, 6, or 8. The pattern for even numbers can be described as
\d*\[02468\]

For example,

"123" matches "\d*\[02468\]"

but

"122" does not match "\d*\[02468\]".

**Example 3:** The pattern for telephone numbers is (xxx) xxx-xxxx, where x is a digit and the first digit cannot be zero. A regular expression for telephone numbers can be described as
\(\((1-9)\d\{2\}\)\ \d\{3\}-\d\{4\}\)
Note that the parentheses symbols ( and ) are special characters in a regular expression for grouping patterns. To represent a literal ( or ) in a regular expression, you have to use \( ( and \)\).

For example,

"(912) 921-2728" matches \(\text{\textbackslash{}\textbackslash{}[1-9]\textsl{\textbackslash{}d}2\text{}\textbackslash{}\textbackslash{}\textbackslash{}d}\text{}\textbackslash{}3\text{}-\textbackslash{}\textbackslash{}\textbackslash{}d}\text{}4\)\)

but

"921-2728" does not match \(\text{\textbackslash{}\textbackslash{}[1-9]\text{}\textbackslash{}\textbackslash{}\textbackslash{}d}2\text{}\textbackslash{}\textbackslash{}\textbackslash{}d}\text{}3\text{}-\textbackslash{}\textbackslash{}\textbackslash{}d}\text{}4\)\)

**Example 4:** Suppose the last name consists of at most 25 letters and the first letter is in uppercase. The pattern for a last name can be described as

\([\text{\textbackslash{}A-Z}\text{}\textbackslash{}\text{\textbackslash{}a-zA-Z}]1,24\)

Note that you cannot have arbitrary whitespace in a regular expression. For example, \([\text{\textbackslash{}A-Z}\text{}\textbackslash{}\text{\textbackslash{}a-zA-Z}]1,24\) would be wrong.

For example,

"Smith" matches \([\text{\textbackslash{}A-Z}\text{}\textbackslash{}\text{\textbackslash{}a-zA-Z}]1,24\)\)

but

"Jones123" does not match \([\text{\textbackslash{}A-Z}\text{}\textbackslash{}\text{\textbackslash{}a-zA-Z}]1,24\)\)

**Example 5:** Python identifiers are defined in §2.4, “Identifiers.”

- An identifier is a sequence of characters that consists of letters, digits, underscores (_), and asterisk (*).
- An identifier must start with a letter or an underscore. It cannot start with a digit.

The pattern for identifiers can be described as

\([\text{\textbackslash{}a-zA-Z}_\_][\text{\textbackslash{}w}$]*\)
Example 6: What strings are matched by the regular expression "Welcome to (XHTML|HTML)"? The answer is Welcome to XHTML or Welcome to HTML.

Example 7: What strings are matched by the regular expression ".*"? The answer is any string.

3 The match and search Functions

You can use the re.match and re.search functions to match a string with a pattern. re.match(r, s) returns a match object if the regex r matches at the start of string s. re.search(r, s) returns a match object if the regex r matches anywhere in string s. Listing 1 gives an example of using these functions.

Listing 1 MatchDemo.py

```
import re

regex = "\d{3}-\d{2}-\d{4}"
ssn = input("Enter SSN: ")
match1 = re.match(regex, ssn)

if match1 != None:
    print(ssn, " is a valid SSN")
    print("start position of the matched text is " + str(match1.start()))
    print("start and end position of the matched text is " + str(match1.span()))
else:
    print(ssn, " is not a valid SSN")
```

Sample Output

```
Enter SSN: 4343
4343 is not a valid SSN
```

Sample Output

```
Enter SSN: 434-32-3243
434-32-3243 is a valid SSN
start position of the matched text is 0
start and end position of the matched text is (0, 11)
```

Invoking re.match returns a match object if the string matches the regex pattern at the start of the string. Otherwise, it returns None. The program checks whether if there is a match. If so, it invokes the match object’s start() method to return the start position of the matched text in the string (line 10) and the span() method to return the start and end position of the matched text in a tuple (line 11).
Listing 2 SearchDemo.py

import re

regex = "\d{3}-\d{2}-\d{4}"
text = input("Enter a text: ")
match1 = re.search(regex, text)

if match1 != None:
    print(text, " contains a SSN")
    print("start position of the matched text is " +
          str(match1.start()))
    print("start and end position of the matched text is " +
          str(match1.span()))
else:
    print(text, " does not contain a SSN")

Sample Output
Enter a text: The ssn for Smith is 343-34-3490
The ssn for Smith is 343-34-3490 contains a SSN
start position of the matched text is 21
start and end position of the matched text is (21, 32)

Sample Output
Enter a text: Smith's ssn is 343.34.3434
Smith's ssn is 343.34.3434 does not contain a SSN

Invoking re.search returns a match object if the string matches
the regex pattern anywhere in the string. Otherwise, it returns
None. The program checks whether if there is a match (line 7).
If so, it invokes the match object’s start() method to return
the start position of the matched text in the string (line 10)
and the span() method to return the start and end position
of the matched text in a tuple (line 11).

4 Flags

For the functions in the re module, an optional flag parameter
can be used to specify additional constraints. For example, in
the following statement

    match1 = re.search("a\{3\}", "AaaBe", re.IGNORECASE)

The string "AaaBe" matches the pattern a\{3\} case-insensitive.
But in the following statement

    match1 = re.search("a\{3\}", "AaaBe")

The string "AaaBe" does not match the pattern a\{3\}. 