Python + Regular Expressions

2009

2 min read • 469 words

Themes: Programming

Have you ever needed to parse through large amounts of text looking for a specific pattern? Patterns like "one capital letter followed by three numbers" or "dd/mm/yyyy"? This is known as Pattern Matching. Regular Expressions allow easy syntax for pattern matching

> Regular expressions represent one of computer science's most enduring abstractions, originating from formal language theory in the 1950s. Their ubiquity across programming languages demonstrates how mathematical concepts from automata theory can become practical tools that transcend specific technologies or domains.

.

Whether you're writing a Compiler, Form Validator, Text Editor, Django Project, or Language Translator, Regular Expressions will always prove to be invaluable.

Python has built-in support for Perl-style Regular Expressions

> Perl's influence on regular expression syntax became the de facto standard across programming languages in the 1990s and 2000s. This cross-language standardization exemplifies how particularly elegant implementations can become universal idioms, creating shared knowledge that transfers between different programming ecosystems.

. The included 're' module will give us everything we need:

```
import re
```

Here's a quick example of basic use:

```
import re

string0 = 'Kenneth Reitz is a cool guy!'
regExp = r'kenneth[- ]?reitz'

if re.match(regExp, string0, re.IGNORECASE):
    print "True"
else:
    print "False"
```

This script takes the string 'Kenneth Reitz is a cool guy', and searches for 'kenneth reitz' inside of it. Note the 're.IGNORECASE' flag used here – This tells the function be case-insensitive.

Here's another example:

```
import re

string0 = '10.03.1988'
regExp = r'^\d\d[.]\d\d[.]\d\d\d\d?$'

if re.match(regExp, string0):
    print 'True'
else:
    print 'False'
```

Most of the time, we're going to want to extract the data that matches our query. We can tell Python to show us the data by breaking our expression up into different groups using parentheses '()':

```
import re

string0 = '10.03.1988'
regExp = re.compile('^(\d\d)[.](\d\d)[.](\d\d\d\d)$')
regExpMatches = regExp.match(string0)

if regExpMatches:
    print("Day: %s\nMonth: %s\nYear: %s" % (
        regExpMatches.group(1),
        regExpMatches.group(2),
        regExpMatches.group(3)
    ))
else:
    print("Invalid Date.")
```

When executed, this script parses through our validated date, breaks it down into groups, and prints:

```
Day: 10
Month: 03
Year: 1988
```

The possibilities are limitless

> This enthusiasm reflects the transformative experience many programmers have when first mastering regular expressions. The ability to express complex text patterns concisely represents a cognitive leap from imperative string manipulation to declarative pattern specification, fundamentally changing how one approaches text processing problems.

!

Quick Reference

```
match:   Match a regular expression pattern to the beginning of a string.
search:  Search a string for the presence of a pattern.
sub:     Substitute occurrences of a pattern found in a string
subn:    Same as sub, but also return the number of substitutions made.
split:   Split a string by the occurrences of a pattern.
findall: Find all occurrences of a pattern in a string.
compile: Compile a pattern into a RegexObject.
```

Remember, you can always type `help(re)` (after importing the re module) into the Python interpreter to take a quick look at the module's built-in documentation.
